CENTRO PAULA SOUZA

FATEC SANTO ANDRÉ

Tecnologia em eletrônica automotiva

Daniel Borges da Silva

PESQUISA E DESENVOLVIMENTO DE *CLUSTER* PARA MOTO

Santo André

2025

Daniel Borges da Silva

PESQUISA E DESENVOLVIMENTO DE *CLUSTER* PARA MOTO

Trabalho de Conclusão de Curso apresentado ao curso de tecnologia em eletrônica automotiva da Fatec orientado pelo Prof. Me. Weslley Medeiros Torres como requisito parcial para obtenção do título de tecnólogo em eletrônica automotiva.

Santo André

2025

FICHA CATALOGRÁFICA

S586p
Silva, Daniel Borges da
Pesquisa e desenvolvimento de cluster para moto / Daniel Borges da Silva. - Santo André, 2025. – 110f: il.

Trabalho de Conclusão de Curso – FATEC Santo André.
Curso de Tecnologia em Eletrônica Automotiva, 2025.

Orientador: Prof. Weslley Medeiros Torres

1. Eletrônica. 2. Motocicletas. 3. Cluster digital. 4. Tecnologia. 5. Conversão de sinais. 6. Microcontrolador. 7. Sistemas analógicos. 8. Sistemas digitais. I. Pesquisa e desenvolvimento de cluster para moto.

621.3815

LISTA DE PRESENÇA

Santo André, 28 de junho de 2025.

LISTA DE PRESENÇA REFERENTE À APRESENTAÇÃO DO TRABALHO DE CONCLUSÃO DE CURSO COM O TEMA: **"PESQUISA E DESENVOLVIMENTO DE CLUSTER PARA MOTO"** DOS ALUNOS DO 6º SEMESTRE DESTA U.E.

BANCA

PRESIDENTE:
PROF. WESLLEY MEDEIROS TORRES _____
MEMBROS:
PROF. PAULO TETSUO HOASHI _____

PROF. EDSON CAORU KITANI _____

ALUNOS:

DANIEL BORGES DA SILVA Daniel Borges da Silva

Aos meus queridos pais, que com tantos sacrifícios e gestos silenciosos me ensinaram o valor da educação, da honestidade e da perseverança. Este sonho também é de vocês.

AGRADECIMENTO

Primeiramente o agradecimento a Deus por ser meu guia e me iluminar, dando forças disposição para alcançar meus objetivos.

A família por sempre estar ao meu lado sendo minha base para que tudo fosse possível.

Ao Professor Me. Weslley Medeiros Torres, que esteve comigo durante essa jornada, orientando e incentivando o desenvolvimento do projeto.

A Faculdade de Tecnologia Santo André pelo excelente ambiente acadêmico que proporcionou contato com os melhores professores que de alguma maneira contribuíram para o desenvolvimento deste trabalho.

Professores Marco Aurélio Froes e Carlos Alberto Morioka pelo auxílio na impressão 3D potencializando o resultado do projeto.

Aos amigos de classe que desde o começo acompanharam esta importante etapa.

A todo processo desafiador que me trouxe até aqui mais forte.

“Tudo o que um sonho precisa para ser realizado é alguém que acredite que ele possa ser realizado.”

(Roberto Shinyashiki)

RESUMO

A presente pesquisa visa a análise e implementação de um *cluster* digital para moto, buscando a substituição de sistemas analógicos e mecânicos trazendo novas funcionalidades e conectividade com o mundo digital. Com tal objetivo foi analisado o funcionamento de sistemas para leitura e conversão dos sinais já presentes no *cluster* original para serem interpretados digitalmente, sem necessidade prévia de ajustes e alterações no sistema receptor original da motocicleta. Já se faz bem presente no mundo automotivo a coleta de sinais analógicos e processamento digital, portanto é necessária a implementação de um intermediário que evite alterações no circuito e interprete o sinal. Para tal finalidade utilizaremos o microcontrolador Raspberry PI Pico RP2040 como componente principal e exibiremos as novas informações no *display* TFT ILI9341, utilizando no código a biblioteca LVGL para geração das imagens gráficas, indicadores de velocidade, marcha, farol alto, nível de combustível e odometria. Para melhor implementação do código, utilizaremos a biblioteca FreeRTOS para melhor gerenciamento do código e processamento através de tarefas.

Palavras-Chave: *Cluster* digital. sinais a/d. display tft. raspberry pi pico rp2040.

ABSTRACT

This research aims to analyze and implement a digital *cluster* for motorcycles, seeking to replace analog and mechanical systems by bringing new functionalities and connectivity with the digital world. With this objective, the functioning of systems for reading and converting signals already present in the original *cluster* to be interpreted digitally was analyzed, without the need for prior adjustments and changes to the motorcycle's original receiver system. The collection of analog signals and digital processing is already well known in the automotive world, therefore it is necessary to implement an intermediary that avoids changes to the circuit and makes the signal tangible. For this purpose, we will use the Raspberry PI Pico RP2040 microcontroller as the main component and display the new information on the ILI9341 TFT display, using the LVGL library in the code to generate graphic images, speed, gear, high beam, fuel level and odometry indicators. For better code implementation, we will use the FreeRTOS library for better code management and processing through tasks.

Keywords: Digital *cluster*. a/d signals. tft display. raspberry pi pico rp 2040.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS, ACRÔNIMOS E ABREVIAÇÕES

ABS	Anti-lock Braking System
ACC	Adaptive Cruise Control
ADC	Analog-to-Digital Converter
ASCII	American Standard Code for Information Interchange
CMD	Command Promp
CONTRAN	Conselho Nacional de Trânsito
CS	Chip Select
EEPROM	Electrically-Erasable Programmable Read-Only Memory
ECU	Eletronic Control Unit
ESC	Eletronic Stability Control
FATEC	Faculdade de Tecnologia
FreeRTOS	Free Real-Time Operating System
GND	Ground
GPIO	General Purpose Input/Output
GUI	Graphical User Interface
I2C	Inter-Integrated Circuit
LED	Light-emitting Diode
LVGL	Light and Versatile Graphics Library
MISO	Master-in Slave-Out
MOSI	Master-Out Slave-in
PCB	Printed Circuit Board
PCI	Placa de Circuito Impresso

PWM	Pulse Width Modulation
SDK	Software Development Kit
SPI	Serial Peripheral *Interface*
TFT	Thin Film Transistor
UART	Universal Asynchronous Receiver/Transmitter
VS Code	Visual Studio Code

LISTA DE SIMBOLOS

Ω Ohm

SUMÁRIO

1 INTRODUÇÃO

Com a crescente evolução e popularização de veículos automotores, cada vez mais são atribuídos mais mostradores e controles que possibilitam o monitoramento e personalização do método de direção em tempo real. Com isso, é necessária a adequação de uma *interface* que seja simples, de fácil compreensão e que garanta a segurança da viagem. Para isso, as mais importantes medições são exibidas no que chamamos de painel de instrumentos ou *cluster* automotivo, onde são exibidas a todo momento o *status* de diversos e opcionais medidores.

Entre os medidores mais importantes podemos citar: Medidor de velocidade, regulamentado pelo "regulamento UNECE R39", onde podemos verificar a velocidade em quilômetros por hora ou milhas por hora, relativo ao sistema métrico padrão de cada lugar onde o veículo é comercializado. Além do velocímetro, há também, com variação de modelo para modelo, medidor de nível de combustível, rotação do motor, indicadores luminosos - luz indicadora de direção, farol baixo, alto, milha, neblina - indicador hodômetro parcial e total, temperatura do motor, além de diversas luzes de diagnóstico e ícones de tecnologias particulares que variam de veículo para veículo.

A evolução exponencial da eletrônica trouxe muitas funcionalidades para os veículos além da imensa capacidade de integração entre sistemas. Com isso cada vez mais sistemas mecânicos são controlados eletronicamente, com o painel de instrumentos não seria diferente. Inicialmente, a engenharia era baseada em sistemas puramente mecânicos com poucos componentes elétricos, o que mais tarde seria modificado para controle eletrônico. Apesar da evolução, os veículos com tecnologias mais antigas ainda estão presentes no quadro de veículos em circulação e representam boa porcentagem. As inúmeras funcionalidades atreladas a sistemas eletrônicos, tornam a adaptação desse sistema em veículos mais antigos atrativa, atraindo o comercio cada vez mais para a criação de equipamentos com poucas adaptações e consequente “modernização” desses veículos sendo capazes de reproduzir novas tecnologias, como por exemplo, sistema de mídia, antigamente puramente rádio AM/FM. Atualmente, diversos tipos de conexão como Bluetooth e

reprodução *online*. A modernização de tecnologias antigas é eminente e ecológica, e está ganhando mais espaço no mercado. (Continental)

1.1 Objetivo

Tendo em vista esse cenário, o objetivo geral é a criação de um sistema de *cluster* digital, herdando as mesmas funcionalidades e capacidades do *cluster* original, evitando modificações de sistemas externos, mantendo o mesmo tipo de entradas das medições originais do modelo. Os sinais deverão ser lidos e convertidos internamente caso necessário, bem como o processamento e renderização das imagens a serem exibidas. Para tal objetivo utilizaremos o processador Raspberry PI Pico RP 2040, com auxílio de sensores para conversão de analógicos mecânicos para sinais digitais inteligíveis pelo microprocessador, como sensor hall e conversor analógico para digital para leitura de sinais elétricos de 0 e 12 volts. Como saída para exibição das imagens processadas, utilizaremos um *display* TFT ILI9341, com sistema de comunicação SPI - *Serial Peripheral Interface,* e densidade de 16 bites de cor, contando com uma resolução de 320x240 *pixels*. Em suma, a proposta é a criação de um painel de instrumentos com a mesma função do original sendo processado digitalmente, substituindo o original.

1.2 Motivação

Devido aos grandes benefícios e funcionalidades trazidas pela era eletrônica em tão pouco tempo, temos como consequência considerável quantidade de veículos que não possuem certas tecnologias, que podem ser consideradas comuns atualmente, e que estão em circulação e com uma longa vida útil a frente. Por tal fato se torna interessante investir na adaptação da nova tecnologia nesses veículos, trazendo melhor conectividade e funções que auxiliam e aumentam o conforto.

1.3 Conteúdo

O conteúdo abordado neste trabalho é sobre pesquisa dos fundamentos do painel de instrumentos de veículos, bem como novas tecnologias aplicadas nos modelos atuais. A partir destes conhecimentos, o cluster digital será desenvolvido, utilizando microcontrolador e tela para exibir as informações.

2 DESENVOLVIMENTO

Atualmente, ao observar diversos meios de locomoção nas mais diversas categorias podemos observar que grande parte, ou praticamente todos os meios, possuem indicadores dos mais variados conforme suas respectivas grandezas, os quais ficam posicionados em fácil acesso e boa vista do condutor ou controlador. O conhecido painel de instrumentos ou *cluster*, conjunto em inglês, é um item essencial e muito característico presente em todos os carros, com grande personalização e modelos que variam conforme a montadora de veículos. Na Figura 1 podemos visualizar um típico painel de veículo leve da montadora Chevrolet:

Figura 1 - Painel de instrumentos montadora Chevrolet

Fonte: Chevrolet (2024)

2.1 História do painel de instrumentos

Apesar de ser muito relacionado a veículos automotores, o Painel de Instrumentos tem como suas origens a locomoção a cavalo. De acordo com a Pirelli, a palavra é originária das carroças e carruagens puxadas a cavalo, e tinha como principal objetivo a proteção dos ocupantes contra o arremesso de lama, decorrente de trotes e vibrações das viagens a cavalo. Produzidos em materiais como metal, couro e madeira, os 'painéis', continham a comida e em situações mais nobres era notada a presença de chicotes e rédeas, entre outras necessidades para as viagens na época.

A figura 2 representa uma carruagem onde é possível notar a presença do painel logo na região frontal da carroça.

Figura 2 - Painel em carroças puxadas a cavalo

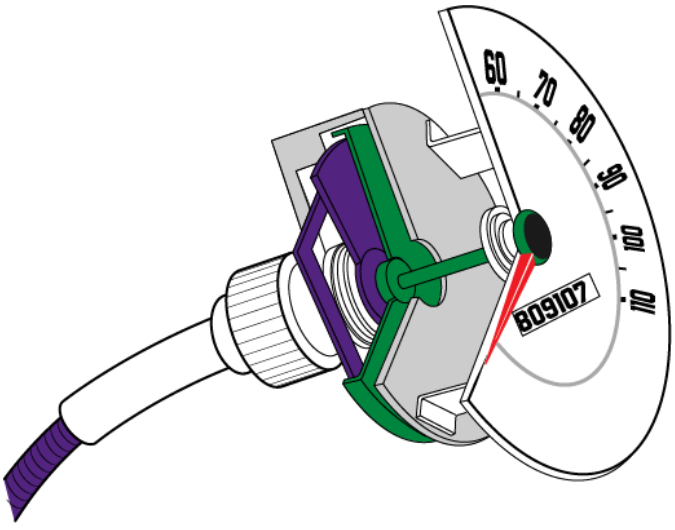
Fonte: Adaptado de BBC (2023)

Com a evolução da tecnologia, os automóveis começaram a surgir e os painéis continuavam presentes. Inicialmente, contavam com um vidro com óleo lubrificante e alavanca de freio, que auxiliavam nas viagens, ainda sem a presença de marcadores e indicadores. Ainda no século XIX, a introdução dos indicadores de velocidade e hodômetro foram inseridos e posteriormente adicionado iluminações para períodos noturnos e de baixa luminosidade, e em alto padrão peças produzidas em latão. Mais

tarde, outros avanços apareceram, como contador de giros e monitoramento de pressão.

De acordo com a Continental, inicialmente os velocímetros não faziam parte dos instrumentos encontrados nos carros, e possuíam versões diferente das que conhecemos hoje. A descoberta do princípio da velocidade da corrente parasita, foi amplamente utilizada na produção dos velocímetros pela interação magnética de atração, possibilitando a criação de um instrumento que era capaz de realizar a medição da velocidade por meio de magnetismo, assunto que será tratado mais à frente. Em 1902, temos como marco o nascimento do velocímetro em Berlim pela empresa Otto Schulze, patenteado como indicador de velocidade de corrente parasita, sendo o a primeira empresa a produzir velocímetros para veículos baseados em uma patente. Na Figura 3 pode-se visualizar uma simples ilustração do funcionamento do velocímetro através do princípio de corrente parasita, onde o imã representado em roxo gera um campo magnético que tende a movimentar o disco metálico verde, interligado com o indicador no mostrador, princípio a ser estudo mais à frente.

Figura 3 - Ilustração velocímetro com princípio de corrente parasita



Fonte: Ractronicos (2020)

Com isso, os veículos automotores começaram a apresentar maiores semelhanças com carruagens, utilizada para viagens mais prolongadas, em que se

passava maior tempo. Outra inovação notável foi a chegada dos autorrádios aquecidos e com válvulas. (Pirelli 2024)

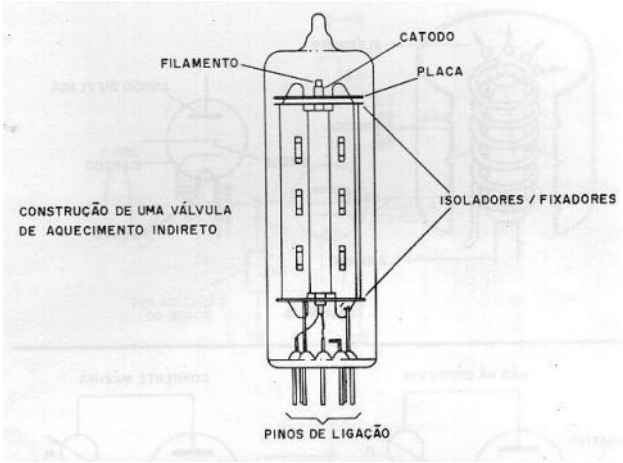
Os auto rádios eram os rádios da época que funcionavam ligados nas baterias de 6V, utilizadas na época e que utilizavam válvulas eletrônicas como mecanismo, que operavam em alta tensão que também eram comuns em uso doméstico, como eletrodoméstico.

Na figura 4 pode-se visualizar um exemplo de auto rádio a válvula do tipo doméstico, com sua aparência vintage e seus mostradores analógicos que informavam a sintonia e na figura 5 as típicas válvulas da década de 30, com algumas de suas peças de construção identificadas.

Figura 4 - Típico rádio à válvula

Fonte: De Volta para o Vinil (2013)

Figura 5 – Esquema genérico de válvula



Fonte: Instituto Newton C. Braga (2021)

Conforme a Continental, com o passar do tempo mais dispositivos eram adicionados ao painel dos veículos, e não demorou muito para que houvesse uma combinação dos mostradores, tornando assim o painel de instrumentos como conhecemos hoje. Melhorias continuaram sendo acrescentadas, como redistribuição dos mostradores para melhor campo de visão do motorista, além de outros indicadores que começaram a fazer parte do *cluster*.

Após a segunda guerra mundial buscou-se renovação que causou forte influência aos carros, onde começaram a inserir a marca ou logotipo do fabricante do veículo no centro dos volantes, cores ao painel de instrumentos e algumas projeções das saídas de ar, autorrádios cromados. A nova indústria dos plásticos rígidos também causou grandes alterações e adaptações ao novo tipo de material, onde eram produzidos alavancas, botões e volantes, e em casos mais populares o isqueiro e cinzeiro. (Pirelli 2024). Na Figura 6 podemos acompanhar como as alterações começaram a transformar o painel do carro.

Figura 6 - Interior Mercury Eight

Fonte: Quatro Rodas (2016)

A chegada da década de 80 trouxe significativas mudanças e avanços tecnológicos que contribuíram para formar o que conhecemos hoje como o painel de instrumentos de um veículo. Nessa época foram introduzidos compartimentos de armazenamento, maiores mudanças para os indicadores que começavam a contar

com novos métodos de obtenção de parâmetros, como velocidade, rotação, etc. Além disso, de acordo com a Continental, foram acrescentados os indicadores luminosos, além do uso dos painéis de instrumentos em barcos e aviações. A era digital contribuiu para maior adequação de funcionalidades a bordo do veículo, a tão conhecida navegação GPS, e posteriormente a ampliação de um sistema de entretenimento. (Pirelli 2024). Trazendo um familiar ar vintage, a Figura 7 traz um típico interior do Miura Saga.

Figura 7 - Interior Miura Saga

Fonte: Quatro Rodas (2016)

Com a chegada da era digital, foram implementados mostradores digitais, por meio de números exibidos em um *display*. Os carros passaram a ter todos os seus indicadores digitais, agora utilizando barras e *Light-emitting diodes* (LEDs), em *displays* baseados em tecnologia LCD. O primeiro modelo foi produzido para a Volkswagen e os típicos painéis podem ser observados em veículos como VW Golf

MK2, VW Jetta e Opel Senator. (Continental). Na Figura 8, observa-se a imagem do painel de instrumentos digital do Opel Kadett.

Figura 8 - Painel de instrumentos digital Opel Kadett

Fonte: WheelsAge (2024)

Os próximos passos da evolução do *cluster* são visualizados atualmente. Maior conectividade, telas sensíveis ao toque e conectividade através da internet. A necessidade da conexão e interconectividade em todos os ambientes se torna mais rotineira e necessária, e com veículos não poderia ser diferente. Para realização do controle há necessidade de adquirir dados para processamento, como informações do ambiente, implementação de sensores e monitores, novas tecnologias que auxiliam o motorista, os quais são processados e por fim são exibidas no painel de instrumentos em tempo real. Aos poucos, é deixado de lado operações básicas de exibição e são implementadas maiores funcionalidades as quais podem ser observadas no painel. Na Figura 9 vemos um belo exemplo de painel repleto de telas e aplicações no modelo Mercedes Benz Classe E. Pode-se perceber que o painel está repleto de telas em uma estruturação basicamente plana com recortes de telas envolvidos por uma superfície preta, ampliando a percepção do tamanho das telas. A esquerda observa-se o painel de instrumentos, exibindo as informações sobre

velocidade do veículo, rotação, luzes de aviso e outras aplicações como simulações do carro, entre outras funcionalidades.

Figura 9 - Painel Mercedes Benz Classe E 2023

Fonte: Estadão (2023)

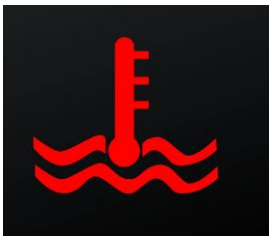
Apesar dos grandes avanços e personalizações do *cluster*, há normas que regem e especificam certos aspectos que devem ser atendidos. Desde 1978, o regulamento UNECE R39 é utilizado para impor requisitos que precisam ser obedecidos para que entre em circulação oferecendo segurança. Dentre eles podemos citar regras de velocidade, a qual não pode ser inferior a real e não pode ultrapassar de 110% da velocidade real. (UNECE).

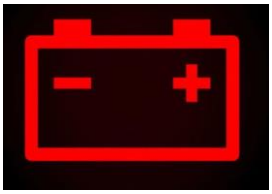
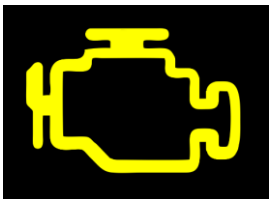
2.2 Controle eletrônico e luzes de aviso

Com os avanços da tecnologia não demorou muito para que fosse implementada a eletrônica embarcada em veículos. O novo recurso trazia consigo a capacidade de pré diagnósticos e alertas dos circuitos que monitoram elementos mecânicos e eletrônicos de controle com apresentação de diversos novos sistemas

de monitoramento. A grande disseminação da tecnologia se espalhou por todos as utilidades do automóvel. Logo, cada parte possuía eletrônica incorporada e não demorou muito até a criação de um meio para que cada circuito de controle se comunicasse. Nesse aspecto, o painel de instrumentos tem o papel de exibir o estado de funcionalidade geral, informando ao condutor que há avarias ou mal funcionamento em algum sistema. A apresentação dessas informações é padronizada e exibida como sinais luminosos no *cluster* através de ícones designados para cada tipo de informação. (Capelli). Há grande quantidade de sinais que todos os carros possuem e outras que variam de carro para carro, veja tabela 1:

Tabela 1 - Luzes de aviso

Luz	**Descrição**	**Imagem**
Luz do óleo do motor	Indica que a pressão do óleo do motor está baixa. Pode ser sinal de falta de óleo ou problema na bomba de óleo.	
Luz do motor	Avaria detectada pelo sistema de diagnóstico de bordo. Requer análise com scanner OBD-II para identificar a falha.	
Luz de temperatura	Alerta de superaquecimento do motor. Pode indicar problemas no sistema de arrefecimento, como falta de líquido de arrefecimento ou falha na ventoinha.	
Luz de abastecimento	Indica que o nível de combustível está baixo e é necessário abastecer.	

Luz do freio	Avisa sobre avarias no sistema de freios. Pode ser problema no fluido de freio ou desgaste das pastilhas.	
Luz da bateria	Sinaliza problemas no sistema elétrico, como baixa tensão ou falha no alternador.	
Luz da injeção	Alerta de falha no sistema de injeção eletrônica do veículo.	

*Alguns símbolos podem sofrer alterações para cada montadora.

Por segurança, as luzes são acesas ao ligar a ignição para que o condutor possa conferir se a lâmpada de determinada luz de diagnóstico está devidamente funcionando ou queimada, o que pode comprometer no alerta. Ao ligar o veículo o carro entra em funcionamento e as luzes são apagadas. Qualquer luz que permanecer acesa ou acender durante o uso e permanecer ligado pode indicar falha. Além das simbologias há diferenças na cor do ícone, elas apresentam o nível de gravidade do problema, em que:

Luzes vermelhas: Indicam um problema grave e deve ser tratado o quanto antes, geralmente com a paralisação em local seguro e desligamento do veículo. Apresentam risco à quebra do veículo caso persista o funcionamento, geralmente relacionados a partes do motor.

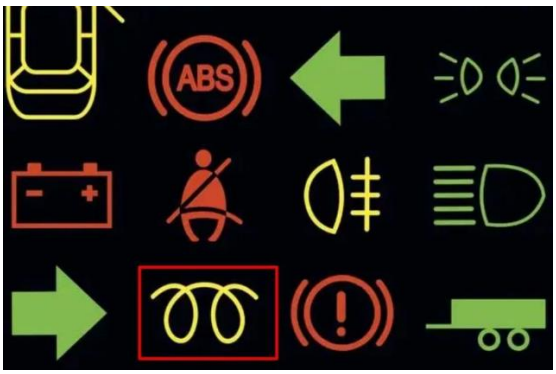
Luzes amarelas ou alaranjadas: Indicam um problema moderado, que não impede a circulação, porém pode se agravar. O carro deve ser levado a um mecânico, em certos casos pode apresentar diferenças no funcionamento.

Luzes azuis, verdes ou brancas: Relacionadas a indicadores de acionamento, indicam se uma função está em funcionamento.

Além dessas luzes indicativas, contamos também com os indicadores do estado do farol e das luzes de direção. Sua função é indicar o nível do farol, posição, baixo ou alto, indicado na cor verde para posição e baixo e azul para o caso de farol alto, em alguns casos, a luz de posição e farol baixo pode apresentar a coloração amarela. As luzes de direção podem ser exibidas por uma única luz indicadora, onde apenas indica se a luz de direção está acesa, sem especificação de qual lado, e o modo de duas vias, onde é possível verificar para qual dos lados está ativa. Sua indicação luminosa geralmente é na cor verde (Capelli 2010).

Na Figura 10 temos alguns símbolos exibidos no painel. É possível visualizar a objetividade apresentada e cores intuitivas, como a simbologia do cinto de segurança, acionado quando o motorista ou um dos ocupantes não está usando o equipamento de segurança.

Figura 10 - Simbologia das luzes de aviso



Fonte: CNN (2021)

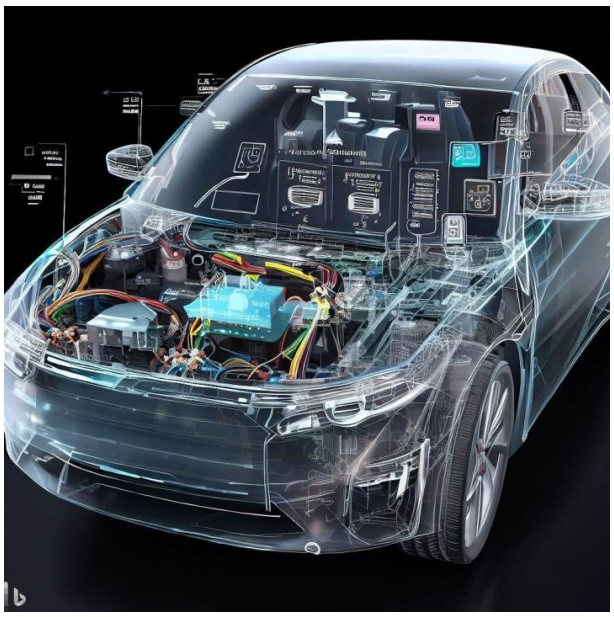
2.3 Hardware de processamento

É inegável que os grandes avanços tecnológicos implementados nos veículos em todos os aspectos melhoraram e tornaram-na uma máquina tão confiável e segura do jeito que conhecemos atualmente. Funcionalidades básicas foram melhoradas, aprimoramentos mecânicos visando segurança e conforto além de uma série de ferramentas que auxiliam no dia-a-dia. Ao falarmos da evolução de veículos em um

todo, podemos destacar o exponencial avanço atrelado a inserção da eletrônica. Cada vez mais o veículo passa ter mais controladores e arquiteturas de intercomunicação e funcionamento baseados em eletrônica digital, possibilitando integração, unificando e processando dados coletados dentro e fora do carro e até mesmo dados em nuvem. (Guimarães 2007).

Por trás de toda tecnologia e *software* de processamento, há um ou mais *hardwares* que fazem tudo funcionar, e quanto maior a tecnologia maior o poder de processamento atrelado ao *hardware*. Os avanços eletrônicos do veículo trouxeram mudanças na arquitetura que facilitam o processamento por meio de segregação de funcionalidades, em outras palavras, podemos dizer que foram separados por aplicação e cada aplicação possuí o que conhecemos como ECU, *eletronic control unit*. É na unidade de controle de determinada aplicação que coletamos e processamos os dados lidos nos sensores através de sinais de tensão. Os dados são interpretamos e traduzidos, em alguns casos já é feito o controle por meio da própria ECU, e quando incorporados em determinada rede de comunicação são informados a outras unidades de controle que possam precisar dos dados. Para o sistema do painel de instrumentos não foi diferente, temos um *hardware* dedicado ao processamento das informações a serem exibidas.

Figura 11 - ECUs em um carro



Fonte: ACHARYA (2023)

A modernização também atribuiu aos automóveis a utilização de telas dos mais variados tipos e tamanhos. O *center stack*, também conhecido como sistema de *infotainment*, foi o principal propulsor a adição das telas nos veículos, uma vez que deixava apenas de informar uma estação de rádio, volume e hora, para se tornar um dispositivo de entretenimento e informação para os ocupantes. As possibilidades são enormes e logo houve grande aceitação da comunidade e a busca por modelos que possuíam mais que um simples rádio multimídia. Vale ressaltar a grande unificação de dispositivos no carro, desde o tipo de mídia a ser reproduzida até sistema GPS, além do grande controle e versatilidade.

Figura 12 - *Center Stack*

Fonte: JVIS

Ao pensar em painéis de instrumentos o mesmo aconteceu, aprimoramentos e estatísticas foram adicionados. O *cluster* que até então era puramente analógico e exibia marcadores de velocidade, conta-giros, combustível e hodômetro, começava a trabalhar juntamente com o carro e conversava entre outras ECUs do automóvel. Com sua finalidade de ser a *interface* entre veículo-condutor, no visor do *cluster* que são exibidas todas as informações que são pertinentes ao motorista, não deixando de lado as informações necessárias, mas adicionando eventualidades e exibições personalizadas. Juntamente com a integração das unidades de controle, os dados

processados possibilitam uma série de aplicações novas, tais como controle de viagem por tempo e distância, através do modo de viagem, consumo de combustível em tempo real e por quilometragem, previsão da autonomia do carro, temperatura ambiente entre outras. *Interface*s de controle e auxilio de direção do carro também são exibidas por meio do painel de instrumentos, a depender de sua disponibilidade, modos de direção, auxílios de controle de velocidade, permanência em faixa além de modos de pilotagem autônoma e semiautônoma, como sistema ADAS - *Advanced Driver-Assistance System*. (Continental).

Portanto, as evoluções e tecnologias que são aplicadas nos sistemas atuais, trouxeram um mundo de novas possibilidades além de aumento da segurança e integração. Ainda assim há sistemas que possuem tecnologias mais antigas e que possuem grande vida útil pela frente. A modernização e agregação de novas funcionalidades em modelos mais antigos torna viável o desenvolvimento de sistemas que sejam capazes de interpretar os sinais para conversão das arquiteturas analógicas para digitais, e como mencionado anteriormente, o objetivo principal deste trabalho.

Para o usuário final são as cores, animações e aplicativos que fazem a *interface* com o usuário. Comandos simples e objetivos por se tratar de uma aplicação veicular fazem com que muitas funções sejam pilotadas por poucos toques e em sistemas mais nobres, até gestos são reconhecidos. Desde trocar de música até ligar mapa de GPS são comandados pelo usuário e exibidas na tela de *infotainment*. Atualmente, já integrada com o veículo, onde é possível até mesmo realizar configurações do automóvel. A grande influência das telas nos veículos trouxe a maior busca por mais telas que consequentemente se espalharam por todos os lugares do carro. Em modelos premium, até os espelhos retrovisores não escaparam e se tornaram compostos por telas e câmera, melhorando ângulo de visão e reduzindo pontos cegos, além de opcionais de mudanças da imagem da câmera e modos de visão noturna. Além do mais, a economia por via de integração dos sistemas para controle em telas se tornou atrativo para as montadoras. Há uma crescente eliminação de botões que ficavam espalhados por todos os lados para dar espaço ao controle por meio da tela principal do carro, geralmente centralizada.

A tendência alcançou também os painéis de instrumentos que passaram a se digitalizar e exibir suas informações por meio de telas. Os grandes indicadores de velocidade, rotação, temperatura e nível de combustível até então exibidos por ponteiros, agora variam conforme o desejo da montadora. Números dinâmicos e barras animadas são os meios mais comuns de exibição de velocidade e rotação do motor, sendo que muitas outras informações são apenas exibidas quando necessário, reduzindo a quantidade de itens que se localizam nas telas e tornando o painel de instrumentos mais objetivo e informativo. (Continental).

2.4 Captação analógica

Ao observar os veículos em circulação é notável que grande parte da amostra observada não se trata de veículos novos, e de acordo com o Correio Braziliense, cerca de apenas 23,5% dos carros em circulação no Brasil possuem até 5 anos de uso. O envelhecimento das frotas no Brasil vem crescendo e já atinge a marca de 10,7 anos, sendo o grande envelhecimento de 2 anos observado entre 2010 e 2022, segundo o relatório do Sindipeças (Sindicato Nacional da Indústria de Componentes para Veículos Automotores). Em veículos comerciais como caminhões e ônibus a média chega a mais de 11 anos de idade média dos veículos em circulação. Com isso, grande parte de tecnologias antigas continuam em circulação, no geral analógicas através de sistemas mecânicos, como cabos. Para melhor entendimento dos sistemas analógicos empregados nesses automóveis é necessário observar as engenharias e como funcionam.

2.4.1 Conta-giros

O conta-giros é um equipamento empregado em diversos tipos de motores e tem como finalidade o monitoramento das faixas de rotações em que o motor atua, auxiliando o condutor nas trocas de marchas, faixas de risco e força do motor. O uso

de conta-giros não se limita apenas a área automotiva e se expande também para a área industrial, principalmente em motores estacionários e equipamentos.

O método de medição analógica mais básico é conhecido como Modo Mecânico. O sistema consiste em um flexível mecânico conectado às engrenagens do motor que transmite a rotação até o painel onde é convertido e exibido por ponteiros de escala. Nesses sistemas o flexível é previamente projetado para evitar curvas para mitigar perdas da transmissão da rotação.

Outro método muito aplicado é a utilização de tacogerador, que funciona com o princípio de eletromagnetismo de motores elétricos, recebendo rotação do motor e levando a energia gerada até o módulo de processamento.

Sistemas mais modernos já utilizam sensores tipo Hall e indutivo, utilizando engrenagens que rotacionam com o giro do motor que é percebido pela variação do campo magnético causado pelos dentes das engrenagens no campo magnético captado pelos sensores. (Capelli 2010)

Figura 13 - Conta Giros

Fonte: AutoPapo (2023)

2.4.2 Velocímetro

Velocímetro é o instrumento que mede a velocidade que determinado objeto se encontra em movimento, amplamente utilizado em meios de transporte. Podem ser de vários tipos, entre eles os mais comuns são os mecânicos (analógicos), eletrônicos, e GPS. Sua unidade de medida pode variar conforme o sistema métrico que o veículo é comercializado, variando entre quilômetros por hora (Km/h) ou milhas por hora (Mph). Há variações conforme aplicações, como barcos e aviões. Sua importância é referente a segurança, regulamento e eficiência da viagem.

Figura 14 - Velocímetro

Fonte: CHAVELO CHAVES E VELOCÍMETROS LTDA.

2.4.3 Nível de combustível

Os medidores de nível de combustível é o dispositivo capaz de obter a informação da quantidade de combustível com base no nível que o combustível está

em relação ao tanque de combustível. Na figura 13, temos um simples e objetivo sistema de nível de combustível. (Guimarães 2007).

Figura 15 - Nível de combustível

Fonte: Instituto combustível Legal (2019)

2.4.4 Hodômetro

Utilizado como referência de distância percorrida pelo veículo desde sua fabricação, no hodômetro é onde fica armazenado a quilometragem percorrida pelo veículo. Diferente das funções de Viagem e Hodômetro parcial que registram a quilometragem de um certo período e podem ser resetadas, o hodômetro não pode ser resetado, por isso é muito utilizado como referência para valor de revenda e manutenções periódicas. Seu funcionamento de contagem pode ser realizado de diversas formas. A partir de um diâmetro de pneu conhecido e com base nas rotações em que rodam, obtemos a distância percorrida pelo veículo em um determinado período, que é contabilizado e exibido no hodômetro, em escala de quilômetros ou milhas a depender do sistema de medidas utilizado no país em que o veículo é

comercializado. Na figura 16, na parte superior temos o hodômetro total e na parte inferior o hodômetro parcial.

Figura 16 - Painel de instrumentos exibindo hodômetros parcial e total

Fonte: Quatro Rodas (2017)

2.5 Aplicações do projeto

Após a análise dos aspectos internos que um sistema veicular completo possui, é necessária a observação e planejamento para que o projeto seja corretamente direcionado. A estrutura de análise se baseará nos dois extremos, como os sinais são originalmente aplicados e como devem sair de modo digital. Com tal finalidade, cabe analisar em que o projeto realmente se baseará. A proposta é implementar um *cluster* digital em uma motocicleta. O modelo escolhido foi uma motocicleta CG TITAN 150

KICK START 2007 da montadora Honda. Na Figura 17 visualizamos o modelo da moto que será utilizada no projeto.

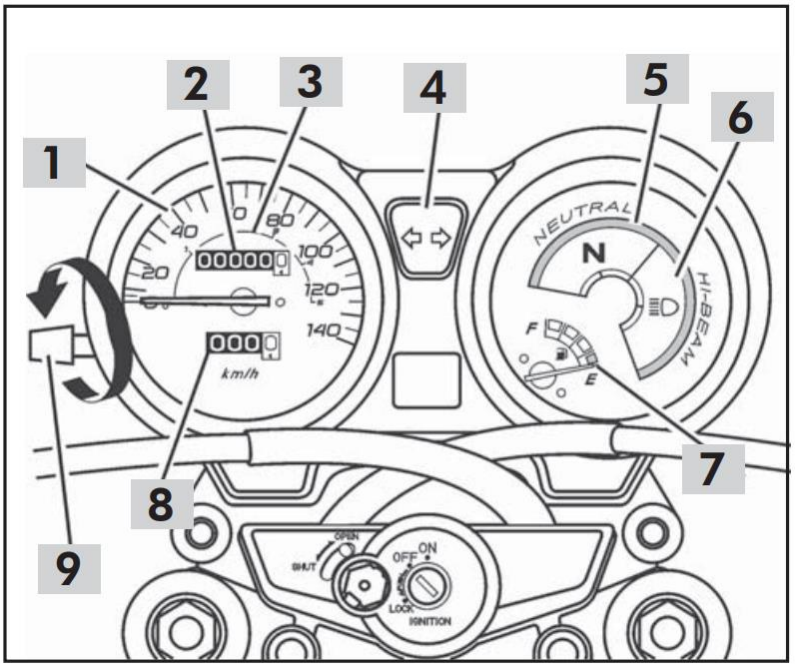
Figura 17 - CG TITAN 150 KICK START 2007

Fonte: MotosBlog (2009)

Por se tratar de um modelo 2007 e de entrada, a motocicleta conta com diversos sistemas mecânicos e analógicos se tornando uma interessante base para a implementação do projeto. Como parte inicial do projeto, cabe analisar os sinais que possuímos no painel de instrumentos original da motocicleta. Ao observar seu painel

de instrumentos podemos ver alguns indicadores, representados na Figura 17, acompanhado da legenda na Figura 18.

Figura 18 - Painel de instrumentos CG TITAN 150 KS 2007



Fonte: Honda (2007)

Figura 19 - Legenda Figura 17

1. Velocímetro: indica a velocidade da motocicleta em km/h.
2. Hodômetro: registra o total de quilômetros percorridos pela motocicleta.
3. Indicador de marcha: indica a velocidade máxima recomendada para cada marcha.
4. Indicador das sinaleiras (verde): pisca quando a sinaleira é ligada.
5. Indicador do ponto morto (verde): acende-se quando a transmissão está em ponto morto.
6. Indicador do farol alto (azul): acende-se quando a luz alta é acionada.
7. Indicador de combustível: indica a quantidade de combustível no tanque.

(Somente CG150 Titan ESD)

8. Hodômetro parcial: registra a quilometragem percorrida por percurso.
9. Botão de retrocesso: zera o hodômetro parcial ao ser girado na direção mostrada.

Fonte: Honda (2007)

Localizado no mostrador à esquerda, o velocímetro da motocicleta é exibido e apresenta marcações de velocidade em quilômetros por hora que são indicados por um ponteiro que rotaciona em seu próprio eixo indicando ao condutor a velocidade atual da motocicleta. Mais ao centro, por não apresentar um conta-giros do motor, vem

apresentando um simples indicativo de marchas do motor por velocidade, que possuem 5 estágios. O modelo de motocicleta utilizado não possui o mostrador de hodômetro parcial.

Localizado na parte central superior, está situada a lâmpada que informa por meio de sinal intermitente, que as luzes indicadoras de direção estão acionadas, sem distinção para qual direção está acionado. Sua coloração é verde.

Já no mostrador localizado à direita, possuímos 3 tipos de mostradores. A luz indicadora de marcha em neutro, que é acionada toda vez que a caixa de marchas se encontra em neutro, emitindo luz verde com a simbologia “N”. Mais abaixo, encontramos a luz indicadora de farol alto. Exibida na cor azul, a luz indicadora de farol alto é ligada quando o motorista aciona o farol alto na motocicleta. Por último, encontramos o mostrador de nível de combustível, que possui 4 estágios que variam de “F”, para tanque de combustível cheio *Full*, a “E” para tanque de combustível vazio, ou *Empty*. Os níveis podem ser observados como; cheio, ¾, meio tanque e reserva, embora seu ponteiro indicador se movimente linearmente.

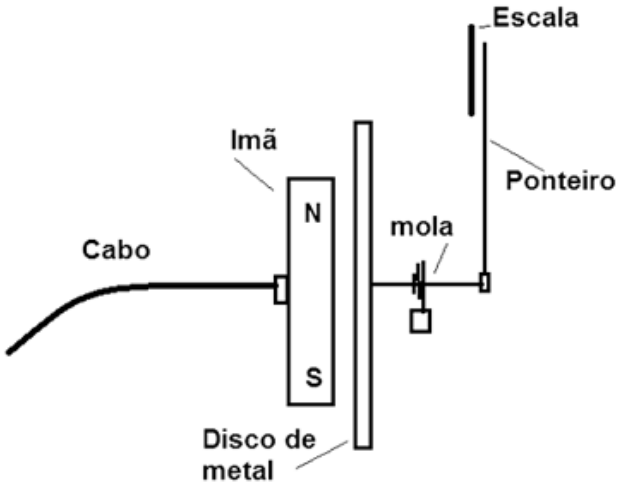
Ao analisar a arquitetura de funcionamento, encontramos simples sistemas analógicos e mecânicos.

2.5.1 Velocímetro e hodômetro

O sistema de velocímetro é do tipo mecânico, assim como o conta-giros mecânico, os velocímetros analógicos mecânicos utilizam o mesmo princípio de flexível ligado a rotação do motor, captado nas engrenagens. Um cabo é conectado à roda dianteira e ao velocímetro, onde há um fio de aço interno com terminal em formato de paralelepípedo com paredes que apresentam ranhuras para melhor aderência mecânica para o movimento. Há uma engenharia de engrenagens internas na roda dianteira que fazem com que o cabo de aço rotacione conforme a roda. Na entrada do velocímetro o sistema funciona com magnetismo. Aqui é utilizado o princípio magnético de corrente parasita. O cabo ou flexível é ligado um imã interno, que ao rotacionar transmite o movimento a um disco de metal na caixa do velocímetro. O disco de metal do velocímetro é diretamente ligado ao ponteiro mostrador. Há uma

mola de retorno e resistência no eixo que liga o disco ao ponteiro. Nota-se, na Figura 20, uma simples ilustração esquemática da arquitetura de funcionamento do mostrador de velocidade e conta-giros.

Figura 20 - Ilustração simples de funcionamento magnético do sistema de velocímetro e conta-giros

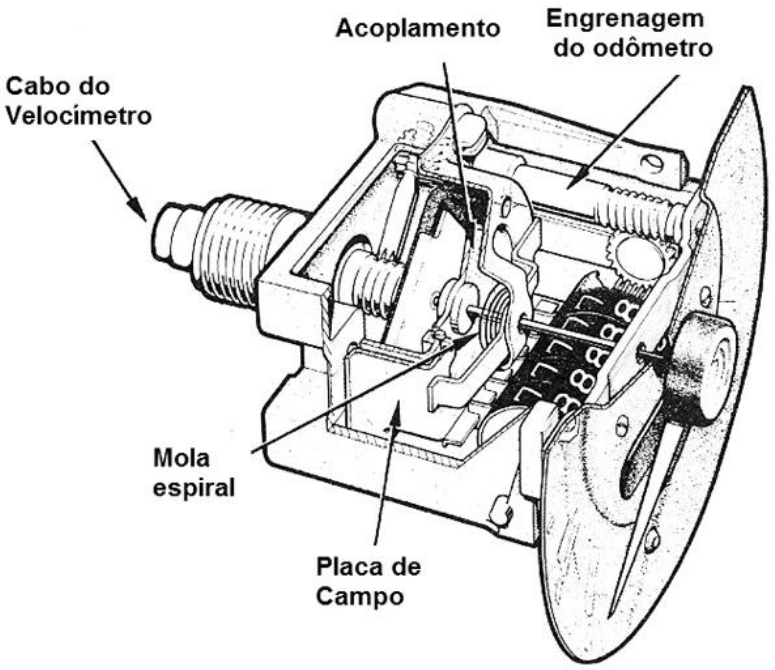


Fonte: Instituto Newton C. Braga (2014)

A Figura 20 traz um simples e objetivo esquema de funcionamento do sistema na vista lateral, portanto na Figura 21, temos um modelo real de velocímetro que utiliza

esse princípio além de ser um modelo similar ao apresentado na motocicleta, apresentando cortes na Base plástica metálica para melhor visualização.

Figura 21 - Ilustração de um sistema de velocímetro real



Fonte: Instituto Newton C. Braga (2014)

No desenvolvimento do velocímetro são realizados cálculos que dão a proporção entre o diâmetro dinâmico da roda e a medida a ser apresentada no painel como indicador de velocidade, com isso também se obtém a distância percorrida, que será exibida no hodômetro, captando a rotação do imã, e levando a rotação diretamente ao mostrador por meio de hastes e engrenagens. Alterações nesse tipo de sistema não são possíveis pela singularidade do mecanismo.

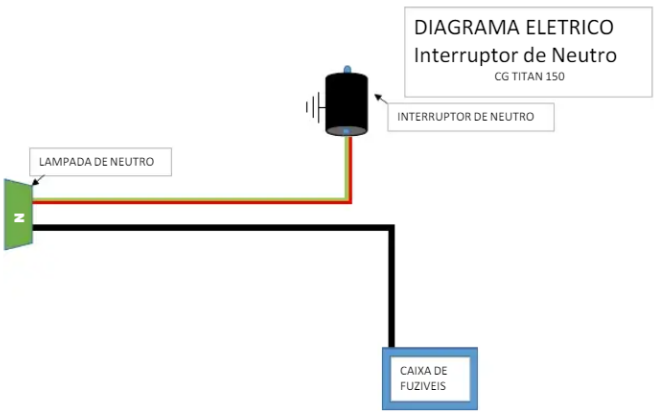
Por tanto, para evitarmos muitas modificações no sistema original, mantem-se o sistema de cabos e é adaptado uma peça similar a uma roda fônica, contendo dentições que mais tarde podem ser lidas por meio de sensor de efeito Hall. Estando conectada a roda, quando o cabo girar irá rotacionar a peça alternando os dentes que

passam próximo ao sensor de efeito Hall, onde é captado a alteração do campo magnético, logo um sinal elétrico é lido.

2.5.2 Indicadores e Iluminação

Para os mostradores de lâmpada, como luz indicadora de neutro, farol alto e setas, o acionamento é realizado com a tensão de 12 Volts que pode ser facilmente adaptada para ser lido como entrada digital com a implementação de um circuito divisor de tensão e grampeador. No sistema de luz de ponto morto ou neutro, o acionamento é realizado por meio de um interruptor mecânico de contato, que ao sincronizar a marcha de ponto morto, gera o contato entre uma palheta situada nas engrenagens e o interruptor, permitindo a passagem da tensão fechando o circuito e acionando a lâmpada. Na figura 21 temos o esquema elétrico da luz de ponto morto.

Figura 22 - Esquema elétrico Luz Neutro

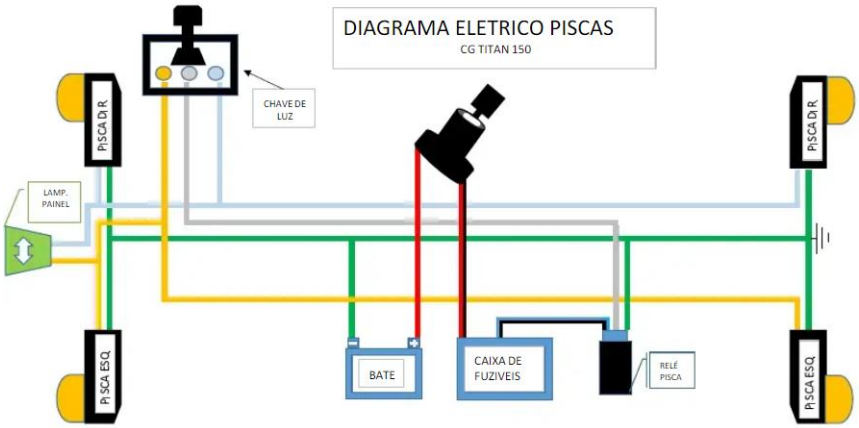


Fonte: Honda (2007)

Para acionar o mostrador das setas direcionais, o circuito utiliza um interruptor acionado pelo motorista, a tensão liberada pelo interruptor alimenta um relé responsável por alternar entre ligado e desligado nas setas, chamado de relé da

sinaleira ou relé de pisca. Conforme a Figura 23, vemos o esquema elétrico do circuito de luzes indicadoras de direção.

Figura 23 - Esquema elétrico Setas

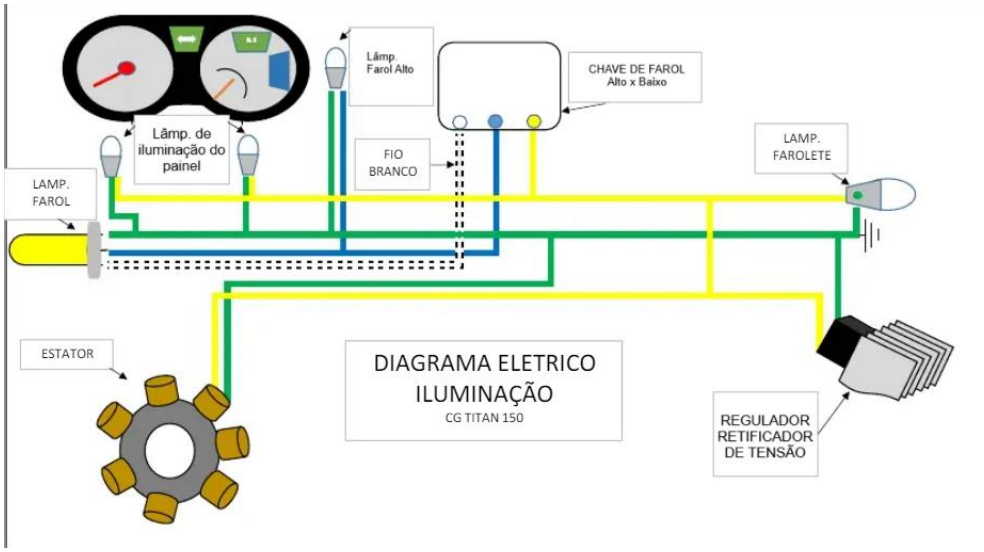


Fonte: Honda (2007)

Para o sistema de iluminação do painel e exibição das luzes de aviso temos um circuito alimentado diretamente pelo estator da motocicleta. Com a energia gerada pelo estator, o Farol e a iluminação de fundo do painel são ligados automaticamente,

e o indicador de farol alto e o farol alto são acionados por meio da chave de farol alto, ou interruptor de farol alto. O circuito de iluminação é feito conforme a Figura 24.

Figura 24 - Esquema elétrico iluminação



Fonte: Honda (2007)

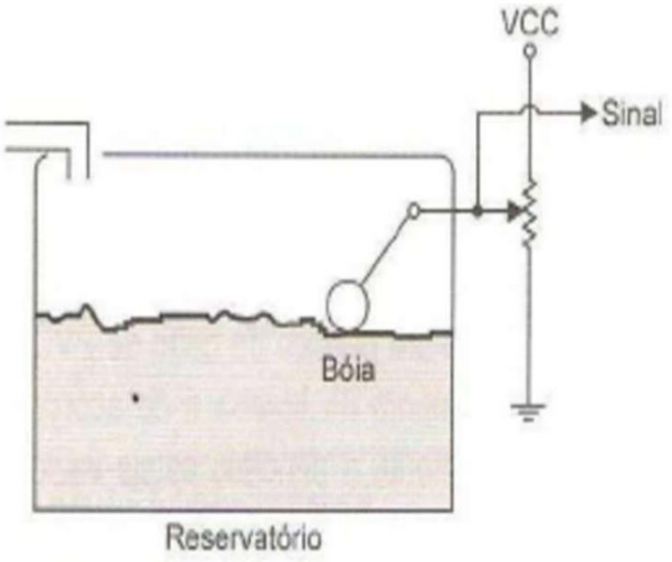
2.6 Nível de combustível

No caso do mostrador de nível de combustível, o funcionamento é por sistema de boia e trilha resistiva localizada dentro do tanque. Neste sistema, um dispositivo é instalado no interior do tanque para realizar a medição, chamado de sensor flutuador potenciométrico, formado por boia, haste e uma trilha resistiva, esse tipo de medidor de nível de combustível utiliza o princípio de resistividade para realizar as medições. Seu funcionamento consiste na flutuação de uma boia sobre o combustível alterando sua altura que por sua vez move uma haste que desliza sobre uma trilha resistiva. Com o valor da resistência alterado, o circuito divisor de tensão gera uma diferença de potencial provinda da medição da tensão sobre um resistor de valor conhecido situado em série com a trilha resistiva do medidor. A tensão lida é proporcional ao nível de combustível presente no tanque. Para exibir a quantidade de combustível, o ponteiro de nível de combustível está posicionado em um eixo, que por sua vez é

rotacionado por meio de bobinas de indução que variam o campo conforme a tensão lida do combustível, alterando a rotação do ponteiro. Neste caso, a entrada por meio da variação de tensão pelo método de divisor de tensão substitui o sistema completamente, uma vez que há a possibilidade da leitura direta da tensão proveniente do circuito de combustível.

Por ser uma aplicação que apresenta grande movimentação, o combustível balança dentro do tanque e o sistema capta as alterações. Para corrigir é implementado sistemas que atenuam essas movimentações e utilizam amostras ou médias extensas para que o nível varie conforme as vibrações e torne a informação confiável, embora a precisão desse sistema seja afetada. Na figura 25, temos um simples e objetivo sistema de nível de combustível. e na Figura 26 o esquema elétrico.

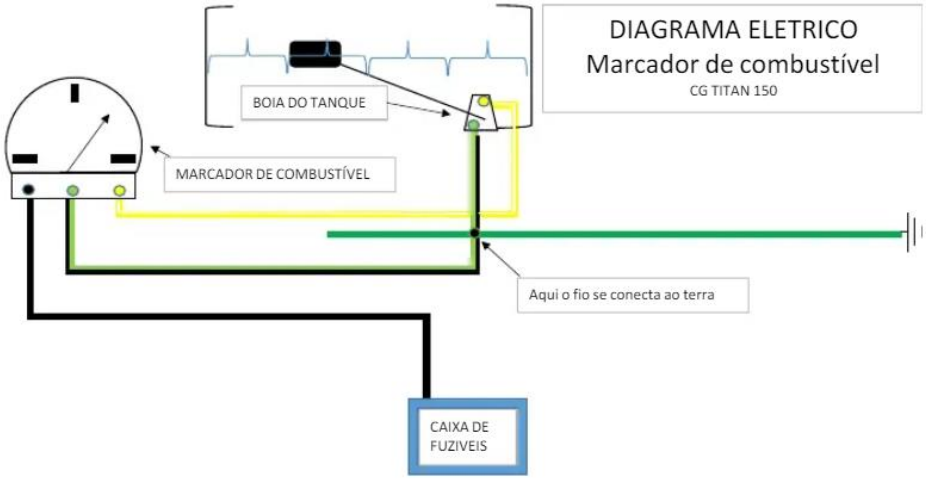
Figura 25 - Sistema de boia de combustível



Fonte: THOMAZINE; ALBUQUERQUE (2006).

Na Figura 26, pode-se ver o esquema elétrico do sistema do medidor de combustível da motocicleta.

Figura 26 - Esquema elétrico nível de combustível



Fonte: Honda (2007)

3 MÉTODOS DE EXECUÇÃO E PROJETO

3.1 Hardware de processamento

Como hardware de processamento, é necessária a escolha de um microcontrolador que possua as entradas necessárias e boa capacidade de processamento para melhor segurança do funcionamento do sistema, trazendo maior de velocidade para as atualizações dinâmicas de velocidade, que é considerado um sistema de segurança. Atrasos na exibição de velocidade podem trazer prejuízos e risco a segurança do condutor e todos ao seu redor.

Devido ao grande poder de processamento, acessibilidade de compra além da utilização didática na Fatec Santo André nas matérias de “Linguagem de programação“, “Microcontroladores”, “Sistemas de conforto e conveniência” e “Tópicos avançados de programação”, foi sugerido o uso do microcontrolador Raspberry Pi Pico, pelo professor Me. Weslley Medeiros Torres. Além familiaridade criada pelo uso nas matérias, futuramente há a possibilidade de uso didático na faculdade pela portabilidade do código e aplicação no conteúdo programático. Na Tabela 2 vamos analisar as características com a finalidade de comparação entre os microcontroladores disponíveis no mercado.

Tabela 2 - Comparativo Microcontroladores

Características	**Raspberry Pi Pico 2040**	**ESP32**	**PIC18F4550**
Arquitetura	ARM Cortex-M0+ (32-bit)	Xtensa Dual-Core (32-bit)	8-bit
Velocidade do Clock	Até 133 MHz	Até 240 MHz	Até 48 MHz
Memória Flash	2 MB	4 MB (externa, dependendo do modelo)	Até 32 kB
RAM	264 kB SRAM	520 kB SRAM	Até 2 kB

EEPROM	Não disponível	Não disponível (pode usar Flash)	Até 256 bytes
GPIOs	26	Até 36 (dependendo do modelo)	Até 35
Conversor A/D	12 bits, 3 canais	12 bits, até 18 canais	10 bits, até 13 canais
Conversor D/A	Não disponível	8 bits, 2 canais	Não disponível
***Interface*s de Comunicação**	SPI, I2C, UART	SPI, I2C, UART, CAN, Ethernet	SPI, I2C, UART, USB
Conectividade	Não disponível	Wi-Fi, Bluetooth	Não disponível
PWM	Até 16 canais	Até 16 canais	Até 10 canais
Consumo de Energia	1,8V-3,3V, baixo consumo em modo sleep	3,3V, eficiente em modo deep sleep	2,0V-5,5V, consumo moderado
Recursos Especiais	PIO (Programmable I/O), RTC	Acelerador de hardware para criptografia	USB 2.0 integrado
Temperatura de Operação	-20°C a 85°C	-40°C a 125°C	-40°C a 85°C
Pacote Físico	DIP, QFN	QFN, WROOM/WROVER, etc.	DIP, QFP
Custo Aproximado	~R$ 40-50	~R$ 30-60 (varia com o modelo)	~R$ 25-40

Ferramentas de Desenvolvimento	MicroPython, C/C++, Arduino IDE	ESP-IDF, Arduino IDE, MicroPython	MPLAB X, XC8
Uso Comum	Projetos gerais, aprendizado	IoT, automação, dispositivos conectados	Aplicações industriais, USB, legado

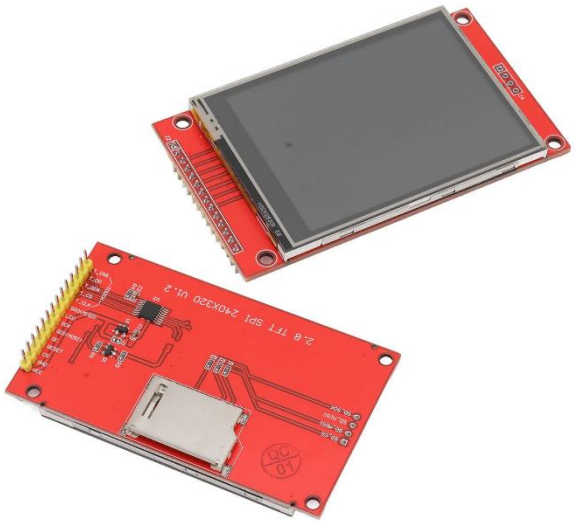
Fonte: Autor (2025)

O microcontrolador Raspberry Pi Pico é um microcontrolador de baixo custo e de alta performance com flexibilidades digitais. Seu processador é um *Dual-core* Arm Cortex M0+, que conta com velocidade de processamento de 133Mhz, Memória *SRAM* 264KB e 2MB de memória tipo *flash*. Suporta USB e possui 26 portas multifunções, dentre elas 2 portas de comunicação tipo SPI, 2 portas de comunicação I2C - *Inter-Integrated Circuit*, 2 portas tipo UART - *Universal Asynchronous Receiver/Transmitter*, 3 portas leitoras de tensão, ADC - *Analog-to-Digital Converter,* com resolução de 12 bits, além de 16 portas de PWM - *Pulse Width Modulation*, controlável. Para aplicação do projeto, como já listado anteriormente, as entradas que o projeto necessita para leitura dos sinais são: 4 entradas digitais e 1 entrada analógica. O tipo de entrada irá variar conforme o sensor utilizado para cada medição, e terá espaço para maior detalhamento ao decorrer do projeto. Para esses requisitos o microcontrolador Raspberry Pi Pico atende e sobram portas para implementações de mais funções para um possível projeto de continuidade.

Em termos de exibição o projeto precisará de um *display* que seja compatível com o microcontrolador e seu tipo de saída. A resolução do *display* impacta diretamente na velocidade do processador por questões de taxa de atualização dos *frames* na tela que acompanham o restante do sistema. Baseado nas dimensões do projeto e acessibilidade para aplicação didática, o *display* ILI9341 cumpre com os requisitos impostos e será utilizado. O *display* ILI9341 é um *display* tipo TFT LCD, com

um único *driver chip* e resolução de 240 por 320 *pixels* e densidade 262 mil cores, exibidos em uma tela de 2.8 polegadas. Na figura 27 vemos o *display*.

Figura 27 - *Display* ILI9341



Fonte: Autor (2025)

Com a estrutura do projeto pré-montada, a parte principal do projeto pode dar continuidade, a programação e processamento dos sinais para exibição no *display*. Para implementação do código será necessária a utilização de bibliotecas que agregam funções ao código, além da economia de tempo na programação e novas arquiteturas de processamento que favorecem no desempenho e gerenciamento do microcontrolador e suas execuções. No quesito de processamento de sinais será utilizado a biblioteca FreeRTOS, que tem como principal função o processamento dos comandos do sistema por meio de tarefas, que são altamente gerenciadas, com métodos de prioridade, tempo, possibilidade de acesso a recursos, além do baixo consumo de recursos do microcontrolador e processamento simétrico e multiprocessador. Sua alta compatibilidade com sistemas e sistema de nuvem também ampliam as possibilidades de aplicação de funções para continuidades do projeto. Para maior refinamento das imagens gráficas a serem exibidas no painel de instrumentos, será utilizado a biblioteca LVGL - *Light and Versatile Graphics Library*, que conta com uma imensa gama de recursos gráficos de exibição, como barras,

gráficos, elementos de texto e reações a eventos que serão utilizados para construção de um *cluster* mais intuitivo e trabalhado. Ambas as bibliotecas possuem código aberto e seu uso é gratuito, possibilitando maior versatilidade na aplicação ao projeto.

3.2 LVGL

A biblioteca LVGL (*LittlevGL, ou Little and Versatile Graphics Library*), é uma biblioteca de código aberto utilizada para gerar *interface*s gráficas de usuário, conhecidas como GUIs, *Graphical User Interface*, amplamente aplicadas em microcontroladores e microprocessadores por ser leve e responsiva. Com a aplicação da biblioteca LVGL no código de programação são inseridas diversas melhorias ao lidar com gráficos, como técnicas de renderização otimizadas além de inúmeros recursos de elementos gráficos pré-definidos, como widgets, além de animações e transições que tornam o resultado final mais confortável ao usuário.

3.3 FreeRTOS

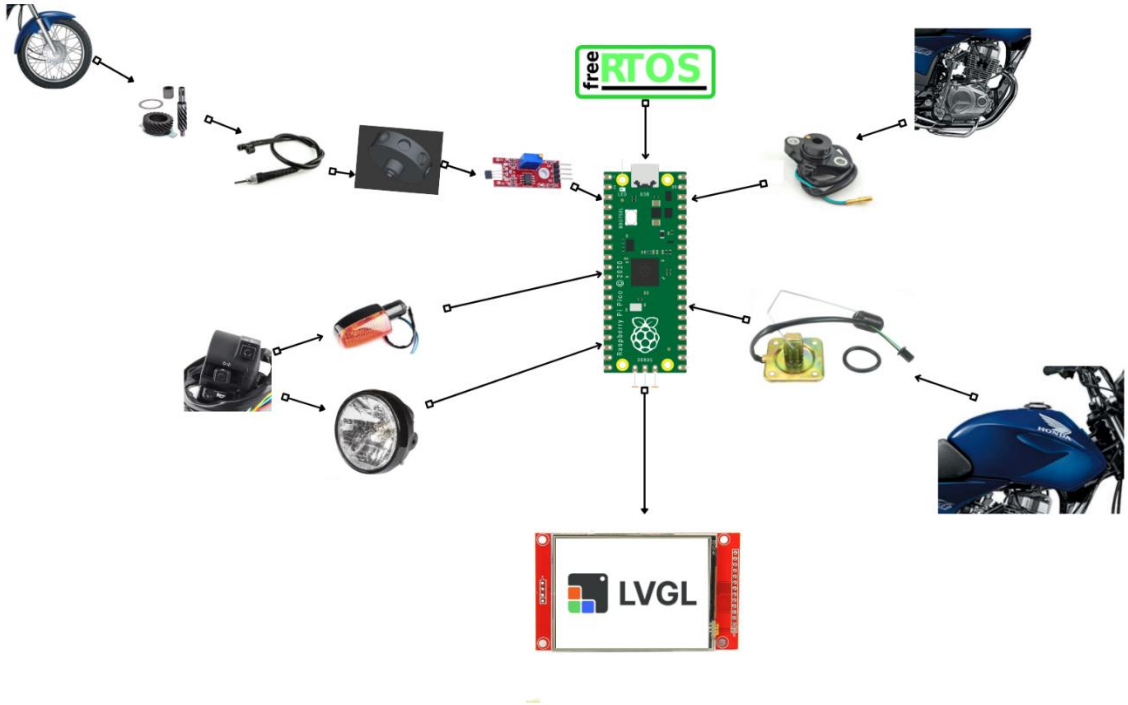
FreeRTOS ou *Free Real-Time Operating System*, é uma biblioteca que traz um sistema operacional de tempo real ao processamento, permitindo melhor gerenciamento dos processos por meio de tarefas otimizando o processamento e resultado. Suas atribuições atribuem ao código funcionalidades como escalonamento, sincronização, comunicação intertarefas, gestão de tempo e memória, através do gerenciamento do código por meio de tarefas, além de economiza e otimização de recursos.

3.4 Aplicação do projeto

Para o início da parte prática do projeto, é necessário que a estruturação seja feita e a base bem desenvolvida. A parte principal do projeto em questão se dá pelo processador, Raspberry Pi Pico 2040, que realizara, por meio de sensores, a leitura dos parâmetros físicos, realizará os cálculos necessários, além do controle do *display*,

bem como a renderização das imagens para que sejam exibidas. O diagrama de blocos da maneira que os sinais serão captados são mostrados na figura 28.

Figura 28 - Diagrama de blocos do funcionamento geral do projeto



Fonte: Autor (2025)

3.5 Programação base da Raspberry Pi Pico RP2040

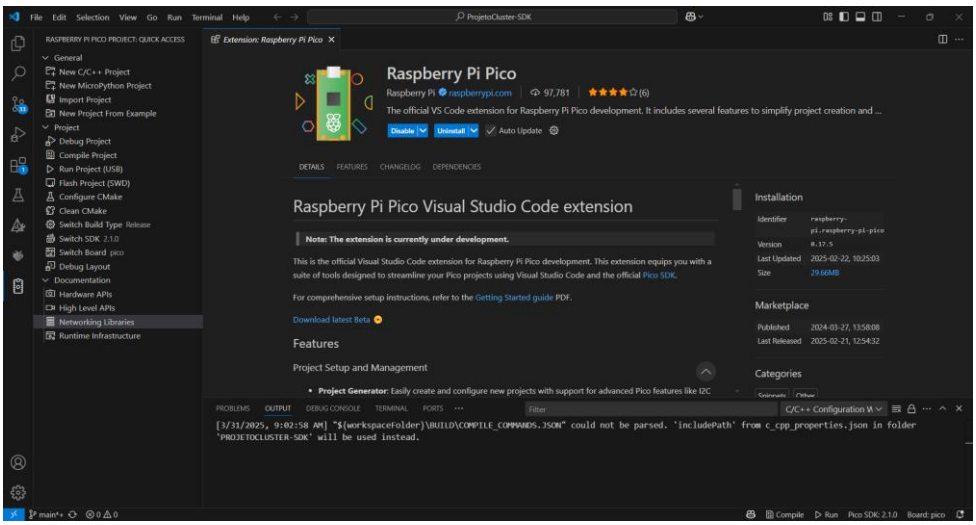
O processador do projeto é a parte mais importante de todo o sistema por ser o componente que comanda todos os periféricos, deve-se ter uma boa lógica de programação para que os comandos rodem sem erros e evitando equívocos e travamentos indesejáveis.

Para que seja possível escrever os comandos em nível de entendimento humano e serem interpretados pelo processador é necessário um compilador. Na documentação disponibilizada pela Raspberry, disponibiliza o compilador, assim como

toda a SDK - *Software Development Kit*. No arquivo de SDK, há o passo-a-passo para utilização do microcontrolador, os primeiros passos, softwares necessários para funcionamento, MicroPython, GCC (compilador), CMake entre outros que podem acrescentar mais facilidades e funcionalidades para o desenvolvimento de projetos. Também é recomendado na SDK, o uso do VS Code – Visual Studio Code, um editor de código-fonte que possui funções que auxiliam no desenvolvimento do código além da integração com extensões que serão utilizadas, CMake, Git, que será explicado mais à frente, e a extensão de SDK da Raspberry Pi Pico.

A extensão de SDK para o VSCode, da Raspberry, apresenta diversas funcionalidades que facilitam e agilizam o processo de desenvolvimento, como a criação de projetos já configurados para a placa, podendo ser gerados em C ou C++, importação de projetos já existentes para o Pico, projetos para exemplo além de opções de depuração, compilador e gravador além de configurações de versão de SDK, CMake e tipo de placa. Na figura 29 temos a tela da extensão do Raspberry.

Figura 29 - Extensão Raspberry Pi Pico VSCODE



Fonte: Autor (2025)

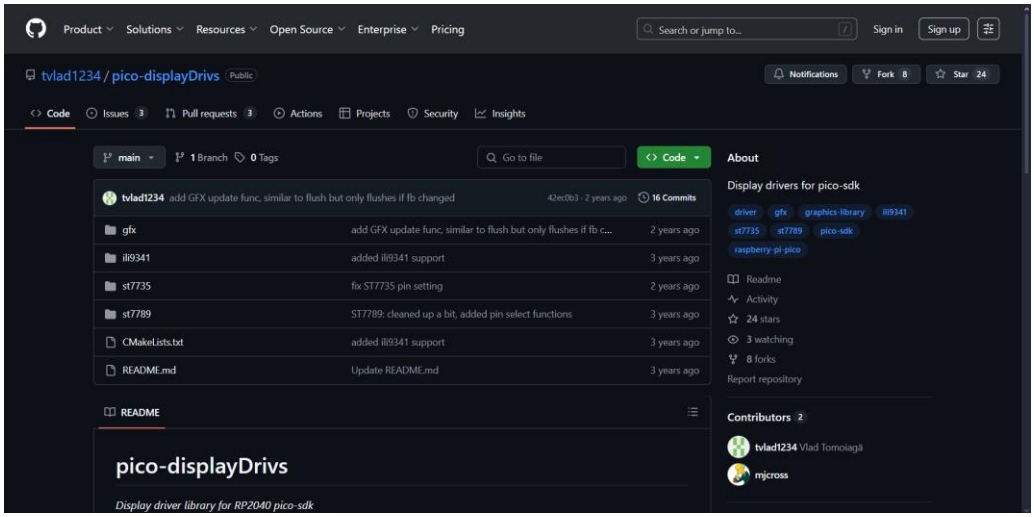
3.6 Adição do GitHub ao projeto

Neste projeto será utilizado o GitHub, uma plataforma online onde são armazenadas diversas bibliotecas, muito utilizado no meio de desenvolvimento pela

versatilidade, capacidade de versionamento em nuvem e múltiplos colaboradores trabalhando no mesmo código. Para melhor organização do projeto, as bibliotecas serão adicionadas em uma pasta chamada “lib”.

Para adição das bibliotecas será utilizado o GitHub, onde acessamos o *link* do repertório da biblioteca dentro do *site* do GitHub. Na Figura 30 pode-se ver o *site* da biblioteca.

Figura 30 - *Site* biblioteca pico-displaydrivs no GitHub



Fonte: Autor (2025)

Para integrar o projeto ao GitHub é necessário abrir um repositório na plataforma. Com tal finalidade, é feito o *download* e instalação do Git Desktop. O aplicativo facilita na clonagem de repositórios entre outras funções. A clonagem pode ser feita diretamente utilizando o terminal interno do VSCode. Após a instalação bem sucedida abrimos o terminal de comandos (CMD), e inserimos os comandos para adição do Git a pasta local do projeto. Localizamos o endereço local do projeto com o comando “cd” seguido pelo endereço. A pasta é localizada, o comando “git init” e inserido, fazendo com que o projeto seja compatível com o GitHub. Um repositório é criado na plataforma *online* do GitHub e os arquivos do projeto carregados. Foi utilizado um repertório particular, e adicionado o professor orientador Weslley Torres como colaborador. Deste modo, o código apenas fica disponível para o dono e os colaboradores.

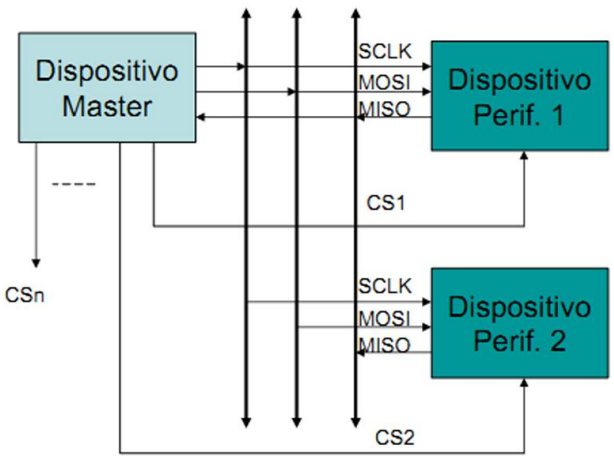
Com a integração do projeto com o GitHub, adiciona-se também a extensão do GitHub ao VSCode, facilitando os *commits*, versões com modificação e sincronismo

com a nuvem. A partir deste ponto a função de clonagem de repositório é possível, e realizada pelo comando no terminal dado por “git clone https://github.com/usuario/repositorio.git”, com a pasta do projeto aberta no terminal.

3.7 Comunicação com o *display* ILI9341

Após a programação inicial no microcontrolador, é necessário a adição do *display* no projeto. O *display* ILI9341, é uma tela TFT LCD com um único driver. Conta com resolução de 240x320 RBG com capacidade de 262 mil cores. Seu meio de comunicação é SPI - *Serial Peripheral Interface*. Criado pela Motorola, o padrão de comunicação SPI é do tipo serial, um dado por vez em sequência, definida por *master/slave* entre dispositivos. Sua frequência de operação varia de 10 kHz a 100MHz utilizando 4 sinais base, SCLK, *Serial Clock*, pulsos que sincronizam os dados, MISO, *Master-in Slave-Out*, MOSI, *Master-Out Slave-in*, como canais de comunicação e CS, *Chip select*, que seleciona o dispositivo a receber a informação. A arquitetura deste sistema é mostrada na figura 31.

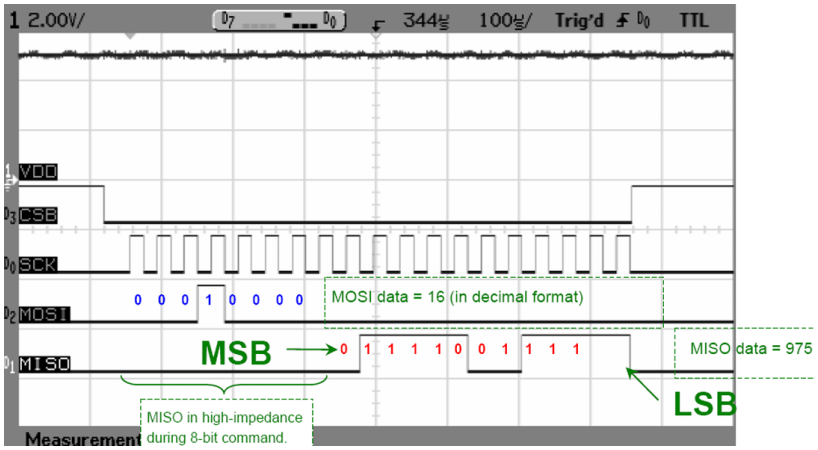
Figura 31 - Arquitetura comunicação SPI



Fonte: Torres, Weslley. 2024. (Slide de aula da disciplina Eletrônica Automotiva – FATEC Santo André).

Na figura 32 pode-se ver como é realizad a comunicação SPI.

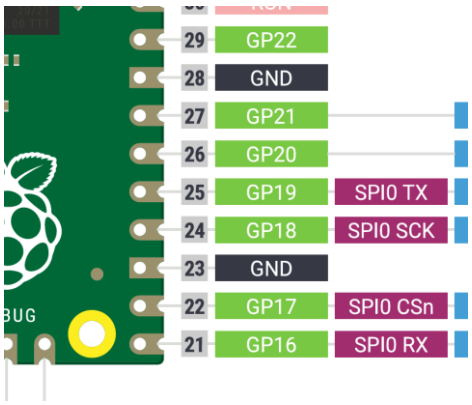
Figura 32 - Imagem osciloscópio comunicação SPI



Fonte: Torres, Weslley. 2024. (Slide de aula da disciplina Eletrônica Automotiva – FATEC Santo André).

Para comunicação com a tela, não será necessário a utilização de pino MISO, apenas o MOSI, comandado pelo processador. A Raspberry Pi Pico 2040 já possui função de comunicação SPI, assim como I2C, facilitando a integração com o *display*. Ao todo a placa possui 2 combinações de porta para comunicação SPI, sendo spi0 e spi1, podendo ser em variados pinos que possuem a capacidade de comunicação. Por padrão os pinos GPIO16 a GPIO20 são utilizados para comunicação SPI, que serão configurados para *interface* com a tela. Pode-se ver na figura 33 os GPIOs do microcontrolador.

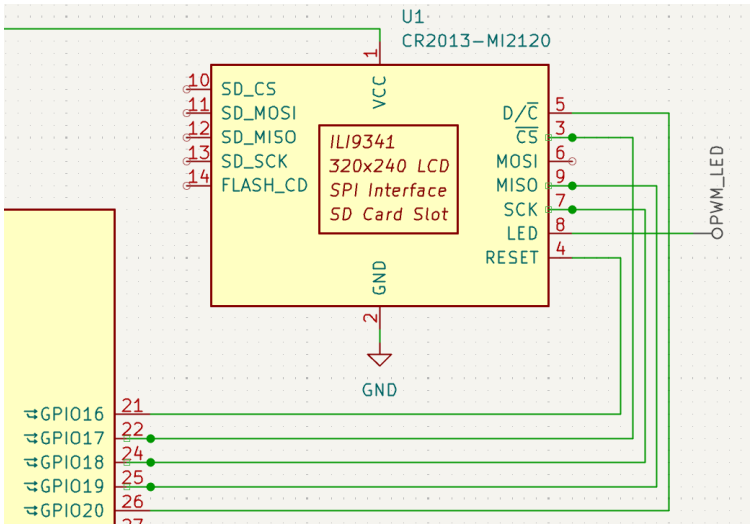
Figura 33 - Caracterização GPIOS Microcontrolador



Fonte: Adaptador de Raspberry Pi Pico

Na figura 34 está o esquema elétrico de ligação do *display*.

Figura 34 - Esquema elétrico *Display*



Fonte: Autor (2025)

Além da ligação dos pinos de comunicação e configuração da placa para realizar a comunicação, também são necessárias funções que realizem de maneira prática o envio das imagens, formadas por cores de *pixel*, ao *Chip* do *display* que irá “colorir” a tela conforme as informações passadas. Para agilização da comunicação com o *display* será utilizado uma biblioteca para tal função.

Para adicionar as bibliotecas e realizar o versionamento do projeto foi utilizado o GitHub. Após uma pesquisa e testes dos códigos de bibliotecas disponibilizados por usuários para realizar a comunicação entre microcontrolador e display, a biblioteca pico-displayDrivs será adicionada ao projeto para realizar a *interface*, por meio desta biblioteca iremos enviar as imagens para o *display*, chamando por suas funções internas. A biblioteca possui diversas funcionalidades e compatibilidade com outros modelos de *display*. Para o *display* ILI9341, existem funções para atualização da tela como *pixel-por-pixel* e *bitmap*, para *bitmap* é passado uma determinada área com coordenadas x e y para definir o início, e o tamanho em x e y para ser desenhado. Geralmente mais rápida e eficaz que o primeiro modo. Há também uma simples e objetiva biblioteca gráfica também interna, a pico-displayDrivs, cujo nome é GFX, que possui algumas funções para desenhar na tela e útil para a validação e teste da comunicação com o *display*.

As configurações são aplicadas a biblioteca para que a comunicação funcione com os pinos conectados ao *display* e a frequência correta. Neste projeto será utilizado 40 MHz, frequência máxima de comunicação SPI suportada pelo *display*, visto que uma boa taxa de exibição é o objetivo.

3.8 Adição do LVGL ao projeto

Após a comunicação com o *display* ser bem sucedida e a biblioteca de *interface* com a tela se mostrar funcional, é preciso adicionar a principal biblioteca de *interface* gráfica, através do LVGL as imagens exibidas no *display* serão geradas. Após o clone do LVGL à pasta, juntamente com a pico-displayDrivs, deve-se adicionar o LVGL ao projeto por meio do arquivo CMake, adicionando o caminho da pasta. De acordo com o LVGL é necessário que seja alterado um arquivo que contém as configurações gerais do LVGL, adequando ao projeto, à placa e ao *display* que será utilizado, o "lv_confg.h". Neste arquivo será selecionado o tipo de cor do *display*, que possui o padrão RGB565, onde 5 *bits* são para cor vermelha, 6 para cor Verde e 5 para a cor azul. A quantidade de memória RAM para ser utilizada pelo LVGL, *buffers* internos, logs de depuração entre outras compatibilidades também são configuradas nesse arquivo. O LVGL traz um arquivo "lv_config_template.h" que possui a explicação de cada configuração que auxiliam na hora de configurar cada componente. De acordo com a documentação disponibilizada no site do LVGL, para um início rápido utilizamos algumas configurações de *buffer*, criação de *display* entre outros, para que o LVGL comece a funcionar no projeto, vemos a documentação na figura 35.

Figura 35 - Documentação *Buffer* LVGL

- Create a display.

```
lv_display_t *display = lv_display_create(MY_DISP_HOR_RES, MY_DISP_VER_RES);
```

- Create a draw buffer: LVGL supports multiple buffering methods. Here you can see how to set up partial buffering (that is render the screen and the changed areas in a smaller buffer). The buffer size can be set freely but 1/10 screen size is a good starting point.

```
/*Declare a buffer for 1/10 screen size*/
#define BYTE_PER_PIXEL (LV_COLOR_FORMAT_GET_SIZE(LV_COLOR_FORMAT_RGB565)) /*will be 2 for RGB565 */
static uint8_t buf1[MY_DISP_HOR_RES * MY_DISP_VER_RES / 10 * BYTE_PER_PIXEL];
lv_display_set_buffers(display, buf1, NULL, sizeof(buf1), LV_DISPLAY_RENDER_MODE_PARTIAL);
/*Initialize the display buffer.*/
```

- Implement and register a function which can copy the rendered image to an area of your display:

```
lv_display_set_flush_cb(display, my_disp_flush);

void my_disp_flush(lv_display_t * disp, const lv_area_t * area, lv_color_t * color_p)
{
    int32_t x, y;
    /*It's a very slow but simple implementation.
     *`set_pixel` needs to be written by you to a set pixel on the screen*/
    for(y = area->y1; y <= area->y2; y++) {
        for(x = area->x1; x <= area->x2; x++) {
            set_pixel(x, y, *color_p);
            color_p++;
        }
    }

    lv_display_flush_ready(disp);         /* Indicate you are ready with the flushing*/
}
```

Fonte: LVGL (2025)

São configuradas as variáveis de definição dentro do código conforme o projeto, como definição horizontal e vertical do *display*. Inicialmente não há alteração no buffer. Na função “my_disp_flush()”, a função de “set_*pixel*(x, y, *color_p);”, é responsável de definir um único *pixel* por vez, essa função deve ser implementada pelo programador, logo utilizaremos a função da biblioteca pico-displayDrivs, que passa um *pixel* por vez, equivalente ao exemplo.

O modo de atualização da tela também é configurado, dado pela função visto na figura 36:

Figura 36 - Função de *draw Buffer* LVGL

Draw buffers

The draw buffers can be set with `lv_display_set_buffers(display, buf1, buf2, buf_size_byte, render_mode)`

- `buf1` a buffer where LVGL can render
- `buf2` a second optional buffer (see more details below)
- `buf_size_byte` size of the buffer(s) in bytes
- `render_mode`
    - `LV_DISPLAY_RENDER_MODE_PARTIAL` Use the buffer(s) to render the screen in smaller parts. This way the buffers can be smaller then the display to save RAM. At least 1/10 screen size buffer(s) are recommended. In `flush_cb` the rendered images needs to be copied to the given area of the display. In this mode if a button is pressed only the button's area will be redrawn.
    - `LV_DISPLAY_RENDER_MODE_DIRECT` The buffer(s) has to be screen sized and LVGL will render into the correct location of the buffer. This way the buffer always contain the whole image. If two buffer are used the rendered areas are automatically copied to the other buffer after flushing. Due to this in `flush_cb` typically only a frame buffer address needs to be changed. If a button is pressed only the button's area will be redrawn.
    - `LV_DISPLAY_RENDER_MODE_FULL` The buffer(s) has to be screen sized and LVGL will always redraw the whole screen even if only 1 *pixel* has been changed. If two screen sized draw buffers are provided, LVGL's display handling works like "traditional" double buffering. This means the `flush_cb` callback only has to update the address of the frame buffer to the `px_map` parameter.

Example:

```
static uint16_t buf[LCD_HOR_RES * LCD_VER_RES / 10];
lv_display_set_buffers(disp, buf, NULL, sizeof(buf), LV_DISPLAY_RENDER_MODE_PARTIAL);
```

Fonte: LVGL (2025)

O `LV_DISPLAY_RENDER_MODE_PARTIAL`, é o modo que o *display* é renderizado, neste caso, utilizando pequenas partes do *display* e então é chamado a função de “my_disp_flush( )”, que envia a área renderizada ao *display* por meio de uma função externa. Por ser atualizado por partes é possível realizar a otimização da atualização da tela por meio da função de bitmap, disponibilizada pela pico-DisplayDrivs, tornando a aplicação mais fluída e imediata. A modificação da função “my_disp_flush” modificada, conforme a Figura 37:

Figura 37 - Função de atualização do display

```
void my_flush_cb_2(lv_display_t * disp, const lv_area_t * area, uint8_t * px_map)
{
    uint16_t size_y = (area->y2 - area->y1) + 1;
    uint16_t size_x = (area->x2 - area->x1) + 1;

    LCD_WriteBitmap(area->x1, area->y1, size_x,size_y, (uint16_t *)px_map);

    /* IMPORTANT!!!
     * Inform LVGL that you are ready with the flushing and buf is not used
anymore*/
    lv_display_flush_ready(disp);
}
```

Fonte: Autor (2025)

A variável interna do LVGL area->x1 e area->y1 possuem as coordenadas iniciais da área a ser renderizada. A variável size_y e size_x formam criadas para armazenar os valores absolutos da área, conforme o *display* atua, convertendo a entrada de lv_area_t para 2 variáveis de uint16_t. O px_map é o dado que possui as cores dos *pixels* a serem enviados para a tela em modo de bitmap, e realizado a conversão para uint16_t por ser o modo de entrada da função. Em suma, é realizado as conversões para o método de entrada da função de *interface* com o *display*.

Testes são realizados para a validação do funcionamento do LVGL, disponibilizado nos exemplos da documentação do LVGL, como visto na figura 38.

Figura 38 - Objeto LVGL slide simples

Fonte: Autor (2025)

Para funcionamento do LVGL é necessário a aplicação de uma *interface* de *ticks*, que gerenciam o tempo e realizam chamadas internas para processamento do LVGL. Os chamados são através de funções que devem ser inseridas dentro de um *loop* principal do código, de alta prioridade, e que repitam em chamadas regulares de tempo em milissegundos. Geralmente, a chamada de função de *ticks* é colocada dentro de um *loop* principal de condição infinita e definido o tempo de chamada, acompanhado por um *delay* que possa garantir a chamada em um tempo regular. Nessa aplicação, será utilizado FreeRTOS, logo a *interface* de *ticks* será gerenciada juntamente com a aplicação do FreeRTOS ao trabalho, juntamente com o restante das aplicações de *interface*, abordadas à frente.

3.9 Adição do FreeRTOS ao projeto

Para adicionar o FreeRTOS Kernel ao projeto é realizado o mesmo passo a passo de clonagem de um repositório do GitHub. Com o repositório devidamente clonado e adicionado como submódulo, dentro da pasta lib, adicionamos a pasta ao

CMake. Seguindo o passo a passo fornecido pelo *site* do FreeRTOS adicionamos as linhas de livraria e configuração conforme a placa utilizada, visto na Figura 39.

Figura 39 - Configurações do CMake para FreeRTOS

```
set(FREERTOS_SRC_DIRECTORY "${CMAKE_CURRENT_SOURCE_DIR}/lib/FreeRTOS-Kernel")

add_library(FreeRTOS INTERFACE
    ${FREERTOS_SRC_DIRECTORY}/event_groups.c
    ${FREERTOS_SRC_DIRECTORY}/list.c
    ${FREERTOS_SRC_DIRECTORY}/queue.c
    ${FREERTOS_SRC_DIRECTORY}/stream_buffer.c
    ${FREERTOS_SRC_DIRECTORY}/tasks.c
    ${FREERTOS_SRC_DIRECTORY}/timers.c
    ${FREERTOS_SRC_DIRECTORY}/portable/MemMang/heap_3.c
    ${FREERTOS_SRC_DIRECTORY}/portable/GCC/ARM_CM0/port.c
)

target_include_directories(FreeRTOS SYSTEM
INTERFACE
    .
    ${FREERTOS_SRC_DIRECTORY}/include
    ${FREERTOS_SRC_DIRECTORY}/portable/GCC/ARM_CM0
)
```

Fonte: Autor (2025)

Assim como o LVGL, o FreeRTOS também possui um arquivo que deve ser configurado para que funcione corretamente conforme o projeto, selecionando tecnologias disponíveis e alternando entre funcionalidades que devem ou não serem executadas no programa. O FreeRTOS também disponibiliza uma versão padrão do arquivo de configuração com a explicação de cada opção apresentada, facilitando na compreensão e assertividade. Para que determinadas funções sejam ativas é necessário que o desenvolvedor adicione as chamadas *Callbacks*, funções de retorno, que executam pelas chamas internas do FreeRTOS, geralmente utilizadas em situações de erro.

Com a biblioteca de operação em tempo real adicionada, suas funcionalidades e aplicabilidade ao projeto devem ser estudadas para melhor aproveitamento.

O FreeRTOS transforma completamente o método em que o sistema trabalha, saindo da maneira convencional, de uma sequência de instruções para configuração e o programa rodando em um *loop* sequencial. Ao invés disso, o Kernel utiliza um

sistema de tarefas, definidas e configuradas no início do programa. Esse aspecto base caracteriza, e é o fundamento para o funcionamento de todo sistema, o gerenciamento de tarefas.

Gerenciamento de tarefas (*tasks*)

O sistema de *tasks* do FreeRTOS consistem na criação de múltiplas funções que são executadas de forma concorrente, com tamanho predefinido e prioridade, em tempo real. Há dois modos de gerenciamento, cooperativo, onde tarefas em andamento são finalizadas para execução de uma nova tarefa de maior prioridade, e o modo preemptivo, onde tarefas são interrompidas para execução de tarefas de maior prioridade.

Filas (*queues*)

Muito utilizadas para comunicação entre tarefas, as filas são pilhas de dados que podem ser enviadas e recebidas entre tarefas de forma segura.

Semáforos e Mutexes (*Semaphores & Mutexes*)

Utilizados para controlar o acesso a determinados recursos, pode ter a função de monitorar quais tarefas acessam o recurso, tornar o acesso exclusivo e sequenciar o acesso.

Temporizadores (*Timers*)

Método para contar o tempo, gerenciado internamente, muito utilizado para execuções por tempo definido.

Grupos de Eventos (*Event Groups*)

Sincronizam tarefas por meio de bits que são ativos por eventos em qualquer parte do programa.

Buffers de Fluxo (*Stream Buffers*)

Também utilizados para comunicação eficiente entre tarefas e interrupções.

Além de todos os recursos citados o FreeRTOS conta com sistema de gerenciamento de memória que otimiza a execução conforme desejado, as "*heaps*", são configuradas no arquivo CMake e possuem 5 variações:

heap_1.c → Alocação fixa, sem desalocação.

heap_2.c → Alocação e desalocação sem fusão de blocos.

heap_3.c → Usa malloc() e free().

heap_4.c → Melhor gerenciamento com fusão de blocos.

heap_5.c → Vários blocos de memória configuráveis.

No projeto será utilizado a *heap_3.c* pelo método de compatibilidade de memória, o uso de *malloc()* reconhecido pela Pico 2040.

A biblioteca também conta com uma portabilidade para diferentes arquiteturas de processador conforme compilador, o modelo compatível com o microcontrolador é o ARMC0, conforme tabela de compatibilidade.

A reserva de memória RAM no arquivo de configuração também é necessária, e não pode exceder a memória total, a memória já reservada pelo LVGL e por variáveis do programa. Utilizando o compilador da extensão da Raspberry Pi Pico, um cálculo de memória já é realizado e quando excedido é informado, como visto na figura 40.

Figura 40 - Erro de *overflow* de memória RAM

```
Projeto/ProjetoCluster-SDK/build/src/ProjetoCluster.elf" ProjetoCluster.uf2 --family rp2040""
C:/Users/barre/.pico-sdk/toolchain/13_3_Rel1/bin/../lib/gcc/arm-none-eabi/13.3.1/../../../../arm-none-eabi/bin/ld.exe: src\ProjetoCluster.elf sectio
n `.bss' will not fit in region `RAM'
C:/Users/barre/.pico-sdk/toolchain/13_3_Rel1/bin/../lib/gcc/arm-none-eabi/13.3.1/../../../../arm-none-eabi/bin/ld.exe: section .stack1_dummy VMA [20
040000,200407ff] overlaps section .bss VMA [200011c0,2004a0e3]
C:/Users/barre/.pico-sdk/toolchain/13_3_Rel1/bin/../lib/gcc/arm-none-eabi/13.3.1/../../../../arm-none-eabi/bin/ld.exe: region `RAM' overflowed by 41
188 bytes
collect2.exe: error: ld returned 1 exit status
[199/199] Linking C static library lib\lvgl\lib\liblvgl_demos.a
ninja: build stopped: subcommand failed.
```

Fonte: Autor (2025)

No exemplo, o terminal retorna o erro avisando que o tamanho dos arquivos de ".bss" não irão caber na região RAM, e que ultrapassou 41188 *bytes*.

3.10 Pesquisas de Trabalhos

Previamente, o estudo de trabalhos e sistemas funcionais é realizado a fim de obter mecanismos validados, métodos de aplicação e tecnologias que auxiliaram outros trabalhos a serem desenvolvidos.

Para um aspecto geral do desenvolvimento do trabalho, os estudos do trabalho de um ex-aluno Fatec Santo André auxiliaram em na perspectiva da criação de um *software* com controle realizado por eletrônica embarcada e painel *touch screen*, com o título de “Controle Dos Acessórios Elétricos Do Veículo Por Tela *Touch Screen*”, o trabalho mostra como foram criados circuitos e sistema de controle. (Cunha 2023).

Com fundamentação utilizada na verificação de sinais, um estudo dirigido ao TCC “Análise De Desempenho Em Cluster Veicular Utilizando Teoria De Filas”, traz uma visão sobre como funciona sistemas com princípio de filas para obtenção ordenada de dados e mitigar perdas. Prioridades de informações são levadas em consideração, trazendo mais rapidamente informações mais importantes. (Ferreira 2016).

A alternativa para a validação do projeto a ser criado, foi realizado o estudo da monografia sobre “Teste e Validação de um Painel Digital Automóvel usando Visão Computacional”, explana procedimentos utilizados na automação de testes de *cluster*, trazendo os requisitos necessários que cada painel de instrumentos deve apresentar para que esteja em pleno funcionamento. (Teixeira 2018).

Para estruturação da base plástica, os princípios utilizados foram apresentados no trabalho “O painel automotivo – como o redesign desta peça pode ajudar na condução e visibilidade externa”, mostrando maneiras que aprimoram a estrutura e posicionamento dos itens para maior comodidade e segurança para viagens. (Cavaler 2019).

Todas as pesquisas e resultados direcionaram o desenvolvimento do painel de instrumentos, trazendo maior assertividade com aplicações funcionais.

3.11 *Interface* e sinais do Cluster

Após estabelecida as bibliotecas base, é possível prosseguir com o desenvolvimento da *interface* do usuário, parte visual, e mecanismo de processamento e atualização dos mostradores que serão exibidos. Para todos os indicadores serão utilizadas variáveis em formato decimal para que seja feita a *interface* entre as leituras dos sensores e serem passadas para os objetos do LVGL para exibição no *display*.

3.11.1 Velocímetro

O mostrador de velocidade da motocicleta é exibido em formato numérico simples e com um indicador do sistema métrico utilizado. No LVGL, o tipo de objeto utilizado para textos é chamado de *label*, onde é criado uma caixa de texto e passado as informações de texto em modo de *string*, ou *const char*, porém como a velocidade não é um texto constante será utilizado uma ferramenta do próprio LVGL que converte outros valores, decimais, para serem exibidos como texto. A biblioteca possui uma fonte nativa, Montserrat. Porém limitada no quesito de tamanho, dado em *pixels*, tornando os textos pequenos ao serem aplicados a um *display* de 2.8 polegadas. No *Google Fonts*, a fonte escolhida foi Changa, no tamanho de 80 *pixels* de altura, com *design* diferenciado que dará um toque especial no *Cluster*. A fonte é convertida por meio de um conversor específico do LVGL disponível no site, gerando um arquivo ‘.c’, já selecionando quais caracteres serão utilizados para poupar memória, no caso de 0 a 9. A fonte é declarada nos arquivos internos conforme instruído.

O indicador do sistema métrico é possível utilizar a fonte nativa pois terá tamanho menor ao indicador velocímetro. Resultado visto na figura 41.

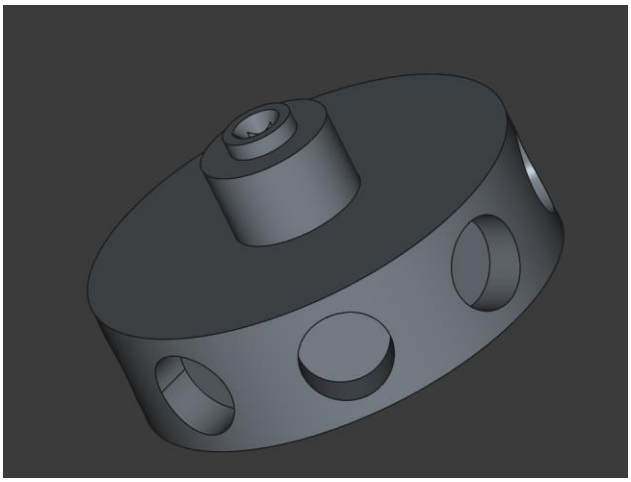
Figura 41 - Velocímetro

Fonte: Autor (2025)

A fonte do sinal de velocidade tem como fonte o movimento da roda dianteira da motocicleta, transmitida por um cabo até o painel de instrumentos. Uma base em plástico é criada para posicionamento de imãs de neodímio, que rotacionarão juntamente com a roda. Para a percepção da rotação dos imãs é utilizado um sensor de campo magnético de tipo Hall, visto que ao rotacionar há variação de campo magnético que é captado pelo sensor. O sensor utilizado foi a placa KY-024, que já possui circuitos para condicionamento do sinal que pode ser utilizado o sinal analógico, onde é mostrado a intensidade do campo, e digital, mostrando se há ou não campo magnético. O modo digital é optado para contagem precisa de quantos imãs irão passar pelo sensor em determinado período de tempo. No código foi utilizado uma interrupção de mudança de borda, realizando a contagem de bordas para definição de quantos imãs passaram. Uma tarefa é responsável por captar essa contagem a cada 500 milissegundos, realizando a conversão da distância com relação as medidas do pneu. A função é responsável por gerar a velocidade para o *label* de velocidade, além de armazenar pequenas contagem de distância que serão utilizadas

para incremento do hodômetro total e parcial. O modelo 3D no FreeCad é exibido na figura 42.

Figura 42 - Base imã FreeCad



Fonte: Autor (2025)

3.11.2 Nível de combustível

Para o nível de combustível será utilizado um objeto de barra, *slide*. No LVGL, as barras podem conter estilos que modificam a aparência, além de animação para alteração do valor apresentado pela barra. Para o nível de combustível o estilo escolhido contém uma borda externa com cantos arredondados passando a percepção da amplitude do tanque de combustível. Se tratando da barra principal, fica menor e exibe o nível atual do combustível.

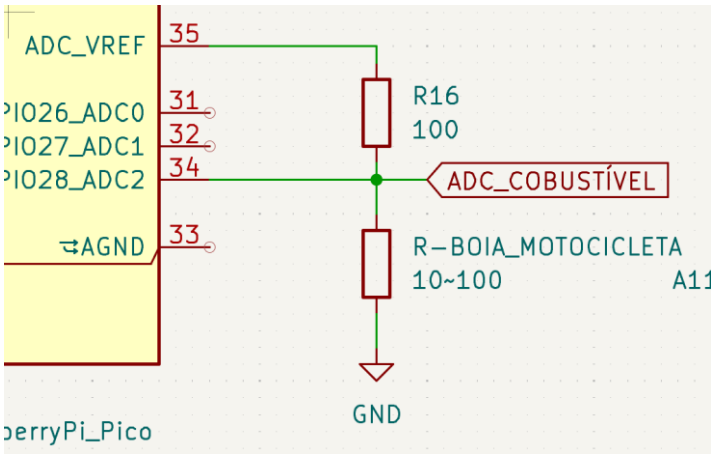
O medidor de combustível fica localizado no lado direito da tela, como mostrado na figura 43.

Figura 43 - Medidor de combustível

Fonte: Autor (2025)

No âmbito de captação do sinal de combustível, uma trilha resistiva tem sua resistência alterada proporcionalmente ao nível de combustível no tanque. O microcontrolador possui uma porta de entrada do tipo ADC, Conversor Analógico-Digital, onde é captada a tensão de entrada. Tendo em conhecimento a variação da resistência, foi colocado um resistor para formação de um circuito divisor de tensão, e definido tensões para tanque cheio e vazio, como visto na figura 44.

Figura 44 - Esquema elétrico do nível de combustível



Fonte: Autor (2025)

3.11.3 Mostradores Luminosos

Os mostradores luminosos informam ao piloto quando a motocicleta está com a marcha em neutro, farol alto ativado e se as setas direcionais estão ativadas. O objeto utilizado é o *LED*, um para cada tipo de mostrador. Suas cores são: Verde para neutro e setas e azul para farol alto, conforme a Resolução CONTRAN nº 970, de 20 de junho de 2022, que estabelece as características e especificações técnicas dos sistemas de sinalização, iluminação e dispositivos de veículos. Esta norma define, entre outras coisas, as cores e significados das luzes do painel, como alertas de emergência, na cor vermelho, alertas de atenção, cor amarela, e luzes indicativas de funções, cores verde, azul ou branca. Os símbolos de farol alto e setas são adicionados por meio do formato de *image*. Para o farol alto foi utilizado o símbolo de farol alto em formato de imagem PNG e utilizado a ferramenta de conversão de imagem para vetor, reconhecido no LVGL, similar ao utilizado no conversor de fonte. O símbolo de setas foi obtido no google *fonts*, no tópico de símbolos e convertido do mesmo modo. Para o neutro foi utilizado o clássico N, em *label*, porém em fonte alternativa, LilitaOne com 36 *pixels* de altura. Na figura 45 temos os mostradores ao lado direito.

Figura 45 - Mostradores

Fonte: Autor (2025)

A obtenção dos sinais dos mostradores é proveniente do sinal positivo enviado no chicote da moto para acionamento das lâmpadas halógenas originais da moto. Reduzindo a tensão 12V para 5V e 0V possibilitando o reconhecimento de sinais digitais pelas portas digitais do RP2040. O circuito utilizado para redução da tensão foi um divisor de tensão com grampeador. O dimensionamento do circuito é baseado na lei de Ohm, onde a tensão de saída é igual a tensão de entrada multiplicada pela resistência equivalente no resistor medido:

$$V_{out} = V_{in} \cdot \frac{R_2}{R_1 + R_2}$$

V_{in}: Tensão de entrada;

V_{out}: Tensão na saída;

R_1: Resistor de base (superior) do divisor de tensão;

R_2: Resistor medido.

Ao arranjar a fórmula para que seja calculado o resistor, temos:

$$R_1 = R_2 \cdot \left(\frac{V_{in}}{V_{out}} - 1 \right)$$

Onde:

V_{in} = ~12V (podendo chegar a 13,8V);

V_{out} = ideal 3,6V (entrada digital);

R_2 = 1000 Ω.

$R_1 = 1000 * \left(\frac{\mathbf{12}}{\mathbf{3,3}} - \mathbf{1} \right)$

R_1 = ~2.640 Ω

$R_1 = 1000 * \left(\frac{13,8}{3,3} - 1\right)$

R_1 = 3.180 Ω

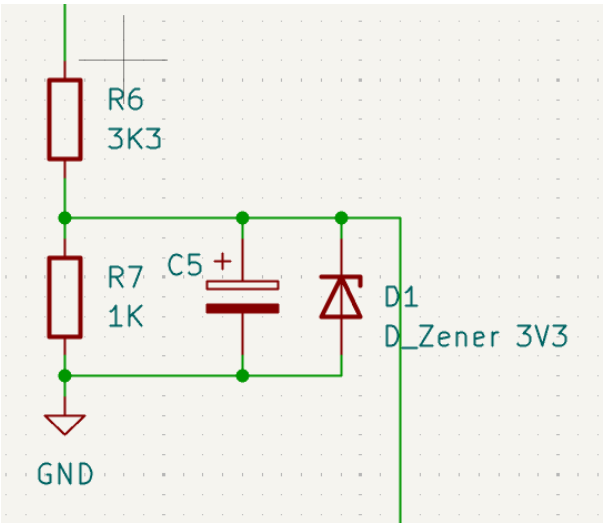
Comercialmente o resistor mais próximo do valor é 3300 Ω, realizando o cálculo na situação de tensão mais alta:

$V_{out} = 13,8 * \left(\frac{1000}{1000+3300}\right)$

V_{out} = ~3,20V

Para maior proteção do circuito é implementado um diodo zener na saída do divisor de tensão, no valor de 3,3V, conforme esquema da figura 46.

Figura 46 - Esquema divisor de tensão com grampeador



Fonte: Autor (2025)

3.11.4 Hodômetro Parcial e Total

Os hodômetros parcial e total são exibidos por objeto de *label*, porém com processamentos diferentes. A base do Hodômetro total tem suas variáveis alteradas com decimais e convertidas para exibição do label, seu incremento é acionado pela tarefa de contagem de bordas, explicada no tópico 6.6.1 Velocímetro. A contagem do hodômetro total vai de 0 a 999.999.999 e concatenado aos caracteres “KM”. Para o Hodômetro parcial foi utilizado um vetor de char, onde inicialmente é definido por “000,0 KM” e incrementado por meio de tabela ASCII, *American Standard Code for Information Interchange*, que possui valores em hexadecimal para cada caractere. Para zerar a contagem do Hodômetro parcial é necessário utilizar a função de “Zerar Hodômetro Parcial” no menu de configurações, mostrado na figura 47.

Figura 47 - Hodômetros

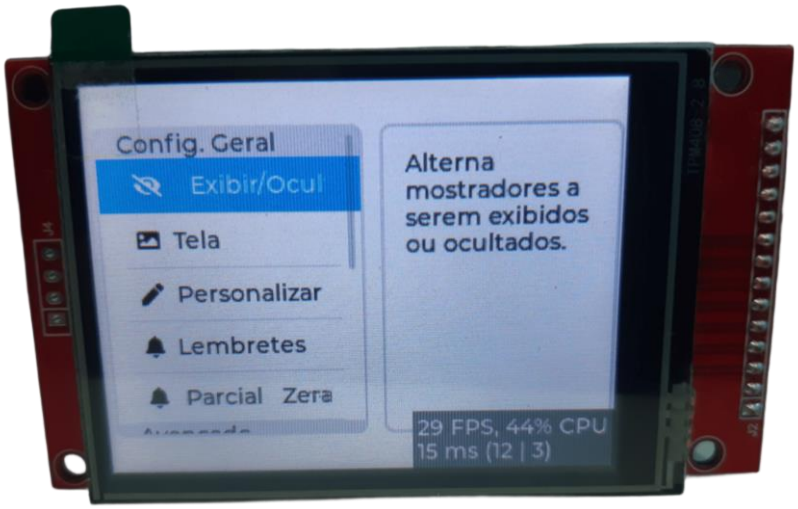
Fonte: Autor (2025)

3.11.5 Menu de Configurações

Um menu incluindo configurações do sistema e personalizações foi incluído ao projeto a fim de maior interação do Cluster com o usuário e adição de funções especiais. Através do menu de configurações é possível a exibição ou ocultação dos

mostradores, definição de tema claro ou escuro, alteração do nível de brilho da tela, entre outras funções de personalização. Também conta com funcionalidades de lembretes acionados por quilometragem rodada, utilizada para avisar ao condutor o momento para troca de óleo e outros componentes de desgaste, uma vez que configurado. Um aviso de velocidade também é acionado, onde o usuário escolhe em qual velocidade quer que o aviso seja disparado. Também há opções de adaptação do Cluster com o veículo, como alteração do tamanho do pneu para correção de velocidade. O menu de configurações é exibido na figura 48.

Figura 48 - Menu de configurações



Fonte: Autor (2025)

3.11.6 Sistemas adicionais

Para o funcionamento em uma aplicação real é necessário a adição de outras funcionalidades presentes na Raspberry Pico, como o salvamento de dados em uma memória não volátil. Foi reservado um espaço de 4 quilobytes na memória *flash*, memória de programa do microcontrolador, para que fossem armazenados os dados da execução do programa. No entanto, essa memória pode apresentar erros caso a gravação seja realizada no mesmo local diversas vezes. Para amenização de perda de memória, foi separado uma maior partição de memória para que o salvamento dos

dados seja “rotativo”, funcionando como um sistema de slots. 4 quilobytes de memória foram reservados para essa finalidade. O Pico, por padrão, realiza gravações em blocos fixos de 256 bytes, logo o código possui 16 locais, ou *slots*, possíveis para armazenamento dos dados que serão gravados na sequência em ciclo infinito, prologando a vida útil da memória. Os dados são passados como uma *typedef struct*, uma estrutura que possui determinadas variáveis, onde contém todas as alterações realizadas pelo usuário e principalmente, os dados de hodômetro, essenciais para utilidade do Cluster. Ao realizar a verificação do tamanho da memória ocupada pela *struct*, o valor de retorno foi de 96 bytes, logo apenas um bloco de gravação por vez será necessário.

Outro método aplicado para maior vida útil da memória, é a gravação apenas quando a moto é desligada. Para tal finalidade é necessário que o microcontrolador permaneça ligado por alguns milissegundos mesmo após a perda da alimentação. Um capacitor é utilizado para manter o sistema funcionando e a detecção do desligamento da energia é necessária, sendo interpretada como um sinal de entrada. Após o desligamento o sinal digital vai a nível lógico baixo, ativando o sistema de salvamento na memória. Para dimensionamento da capacidade de Farads do capacitor utilizamos a fórmula:

$$C = \frac{I \cdot \Delta t}{\Delta V}$$

C = capacitância (Farads)

I = corrente (Amperes)

ΔT = tempo necessário (segundos)

ΔV = queda de tensão aceitável

Os dados de consumo de acordo com o datasheet são:

Estado do sistema	**Corrente típica (3,3 V)**
Núcleo em funcionamento (133 MHz)	~30 mA a 40 mA
Núcleo em funcionamento (máx. 133 MHz + USB)	~50 mA
Dormência (sleep mode)	~1,3 mA
Dormência profunda (dormant mode)	~0,8 mA
Standby com RAM retida (shutdown)	~0,2 mA a 0,3 mA

Será utilizado o consumo com núcleo em funcionamento. O tempo em que a gravação ocorre utilizando a memória flash interna do microcontrolador:

Operação	**Tamanho**	**Tempo típico**
Apagar setor	4096 bytes	~40 ms
Gravar página	256 bytes	~0.8 a 3 ms

O capacitor é alimentado pela saída do conversor de tensão DC/DC, que converte os 12V da motocicleta em 5V para alimentação pelo VSYS do RP2040. A figura 49 mostra a placa do *Step Down.*

Figura 49 - Regulador de tensão Step Down LM2596

Fonte: Autor (2025)

A tensão de funcionamento do Pi Pico 2040 é 3,3V. Com a alimentação de 5V pelo VSYS o regulador de tensão interno é acionado, regulando a tensão de operação para 5V, aumentando a janela de descarga do capacitor. Utilizando níveis mínimos seguros pode-se chegar a 3,0V de alimentação de entrada.

Logo substituindo os valores na fórmula para obtenção do tamanho do capacitor utilizando margem de segurança, temos:

$$C = \frac{50mA * 5ms}{5-3}$$

C = 125 µF

Para apagar a memória, os valores são maiores, contando com 80mA (máximo), e 50ms (máximo), para execução total:

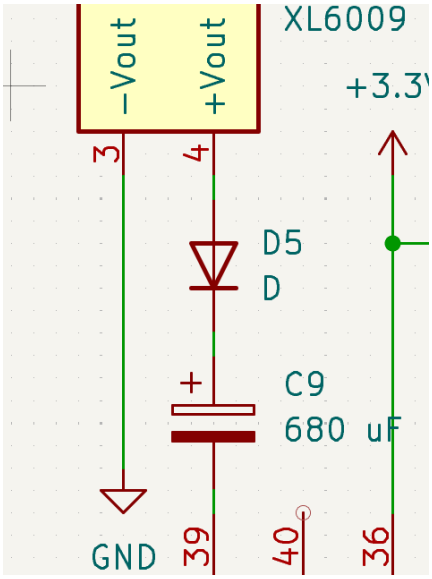
$$C = \frac{80mA * 50ms}{5-3}$$

C = 2000 µF

Pelo alto valor de 2000 Farads de capacitância, o código deverá apenas realizar a gravação ao desligar, e ao ligar, o código verifica como estão os armazenamentos e se necessário apaga o setor completo de 4096 bytes. Visando uma boa margem de segurança foi utilizado um capacitor de 680 µF.

Evitando realimentação, e descarga, do capacitor para a fonte de alimentação é adicionado um diodo ao circuito, isolando a tensão de capacitor apenas para consumo do microcontrolador, como visto no esquema da figura 50.

Figura 50 - Circuito Capacitor Isolado



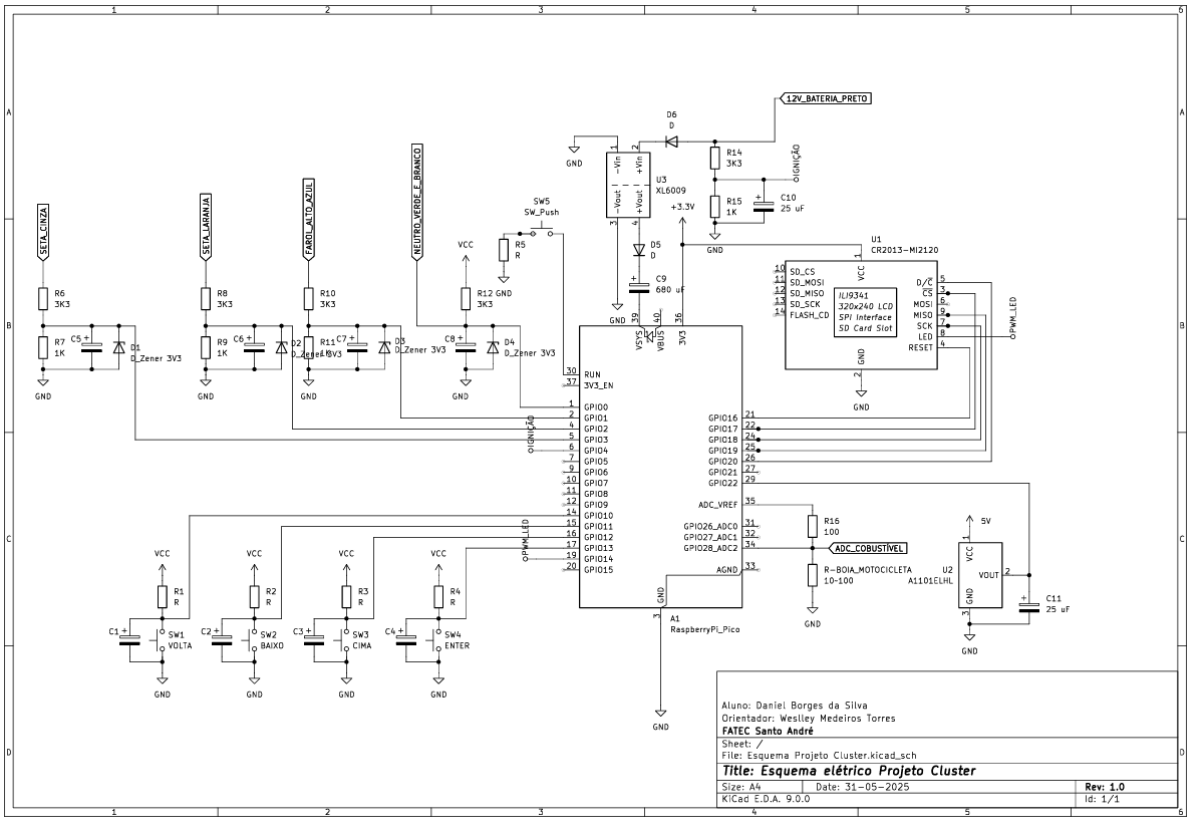
Fonte: Autor (2025)

3.11.7 Montagem e testes

Após os cálculos teóricos para dimensionamento dos componentes dos circuitos, é necessária a montagem da placa para que seja instalada em uma base plástica. Devido à baixa dimensão dos componentes a placa escolhida foi uma PCB universal com ilhas para solda com medidas de 70mmx90mm. Para conexão com a Base plástica foi utilizado conectores tipo bloco. O esquema elétrico foi documentado,

seguindo os valores obtidos nos cálculos de dimensionamento, e o esquema mostrado na figura 51.

Figura 51 - Esquema elétrico

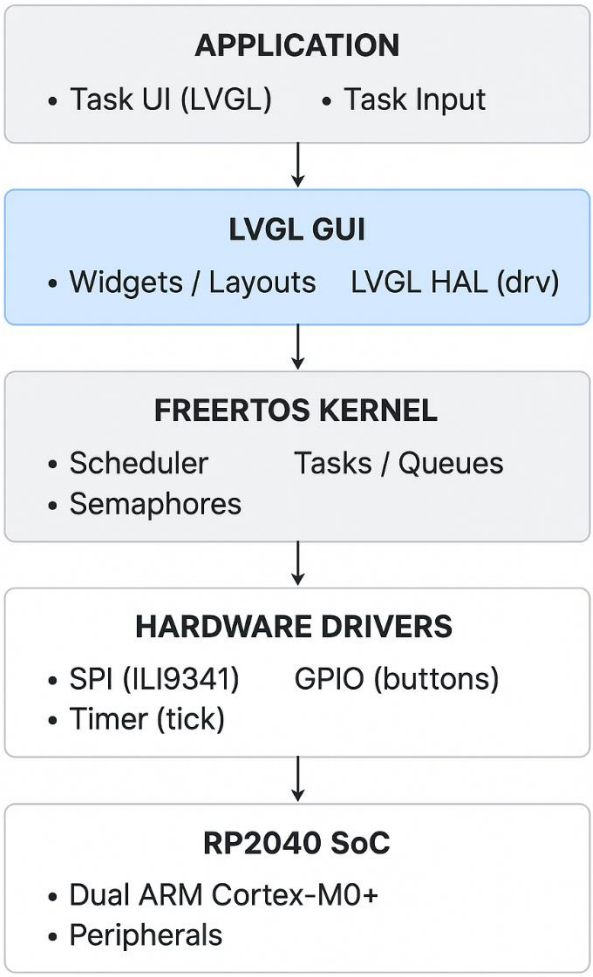


Fonte: Autor (2025)

Em questões de software, a arquitetura de funcionamento é mostrada na figura 52, onde temos o microcontrolador na base, realizando todo o processamento, como leitura de botões e sensores, *timers* nativos e realizando a comunicação SPI com o *display*. A próxima camada é a do sistema operacional, onde se encontram as tarefas que tratam as entradas - baseadas em tempo, processa os dados e gera um resultado de saída. Estes resultados são obtidos pela do LVGL, que transforma os dados em elementos gráficos e cria a imagem a ser exibida no display.

Na camada superficial temos a interação do usuário, como abrir o menu de configurações, gerando *Callbacks* e eventos a serem tratados.

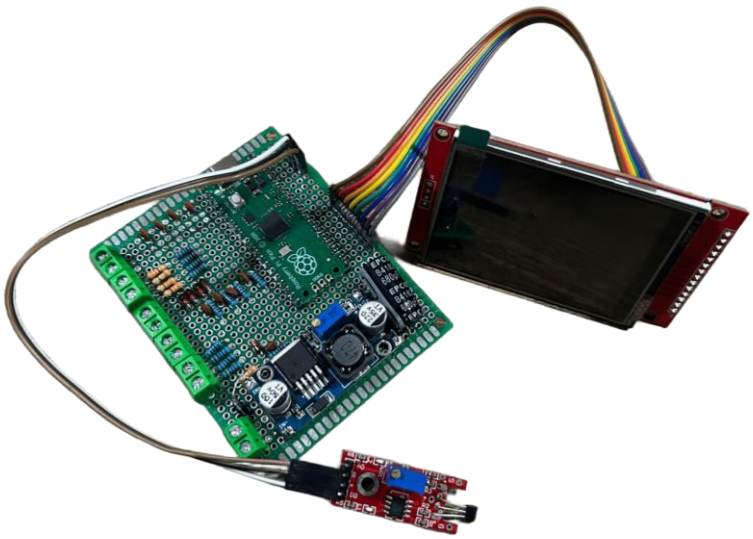
Figura 52 - Diagrama de blocos da arquitetura de software



Fonte: Autor (2025)

Seguindo o esquema elétrico a placa foi montada, com os componentes soldados e posicionados. A figura 53 mostra a placa montada com os componentes, *display* e sensor.

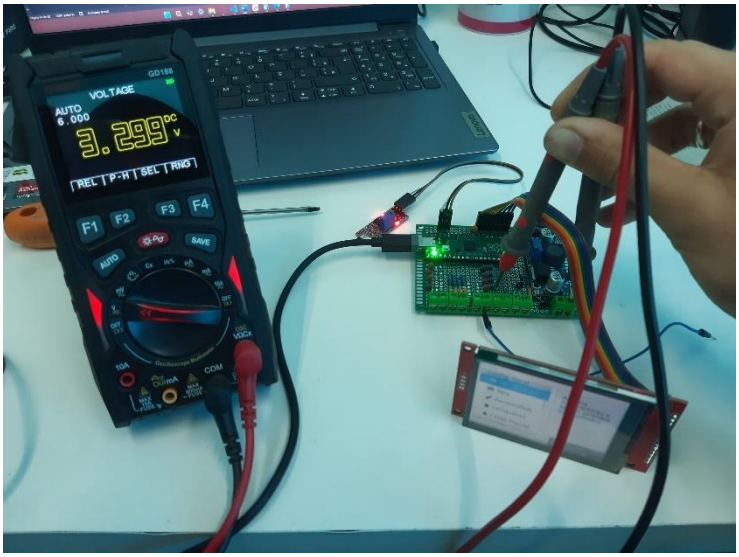
Figura 53 - Placa montada



Fonte: Autor (2025)

Os testes de tensão são realizados a fim de validação e proteção do microcontrolador, como mostrado na figura 54.

Figura 54 - Teste dos circuitos



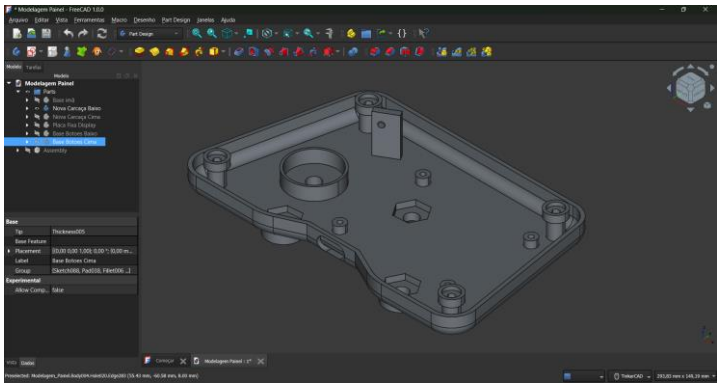
Fonte: Autor (2025)

A placa principal foi montada, todas os tamanhos dos componentes que irão dentro da Base plástica do painel já são definidos, possibilitando o desenvolvimento da caraça.

Com o objetivo de ser utilizado impressão 3D para confecção da Base plástica, a modelagem 3D foi realizada no software FreeCAD, devido a gratuidade do programa, facilidade na modelagem para iniciantes é boa precisão e recursos na construção de modelos. Outras opções como ThinkCAD, Fusion360 e SolidWorks também foram considerados, porém descartados devido à melhores atributos do FreeCAD.

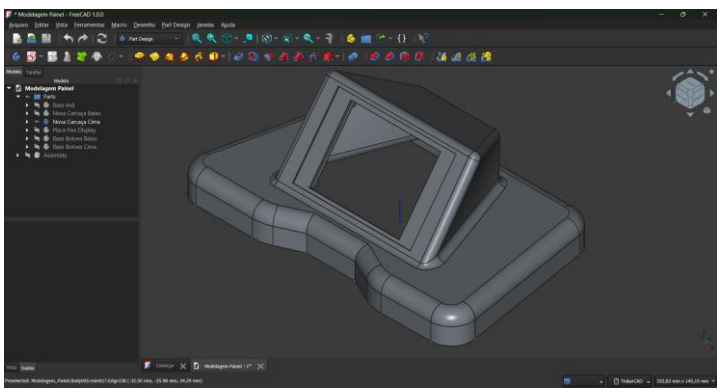
Na modelagem 3D foram criadas diversas peças do projeto, Base plástica Base, Base plástica Superior, suporte de imãs, base de botões e placa auxiliar, seus respectivos modelos 3D são mostrados nas figuras 55, 56, 57, 58 e 59.

Figura 55 - Base plástica de Base



Fonte: Autor (2025)

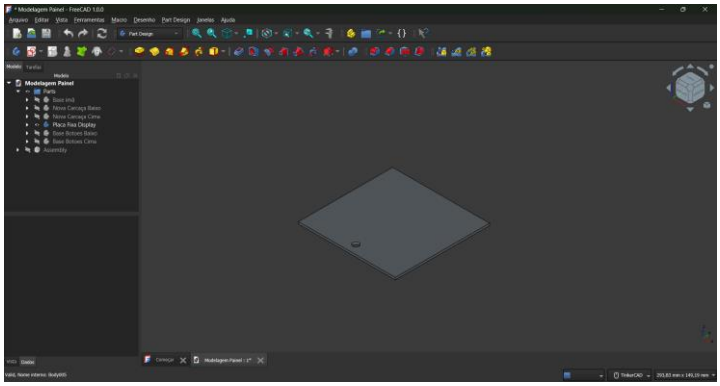
Figura 56 - Base plástica Superior



Fonte: Autor (2025)

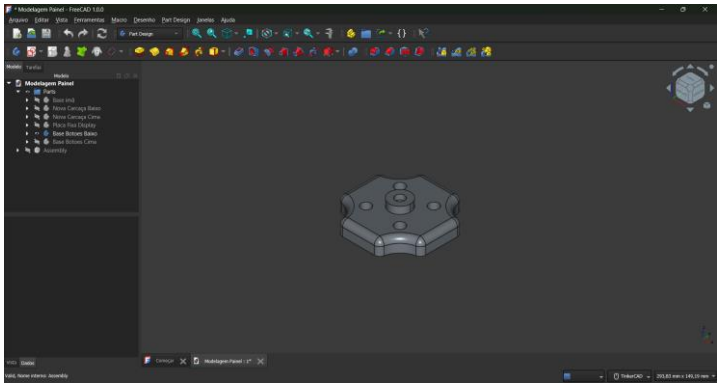
A figura 51 mostra *a placa* de fixação do *display*, enquanto as figuras 52 e 53 a base dos botões.

Figura 57 - Placa de Fixação



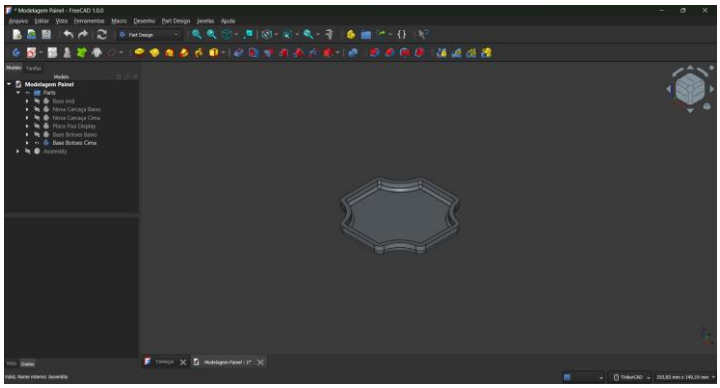
Fonte: Autor (2025)

Figura 58 - Base Botões



Fonte: Autor (2025)

Figura 59 - Base plástica Superior Botões



Fonte: Autor (2025)

Para impressão 3D foi utilizada a impressora 3D da Fatec Santo André, onde gentilmente os professores Marco Aurélio Froes e Carlos Alberto Morioka realizaram a impressão, como mostrado na figura 60.

Figura 60 - Base plástica Impressa

Fonte: Autor (2025)

Com todas as peças e componentes finalizados é realizada a montagem do *cluster*. A figura 61 mostra o resultado da montagem.

Figura 61 - Cluster Montado

Fonte: Autor (2025)

4 TESTES FINAIS

Para teste inicial foi aplicado a sinais simulados com tensões reais de operação da motocicleta para validação final antes da instalação embarcada, onde o projeto respondeu corretamente aos sinais obtidos, sendo exibidos graficamente no *display*. Para validação do nível de combustível é necessária a medição da resistência apresentada na aplicação embarcada. O sinal de velocidade é lido corretamente em simulação pelo sensor após aplicar a aceleração na base dos ímãs.

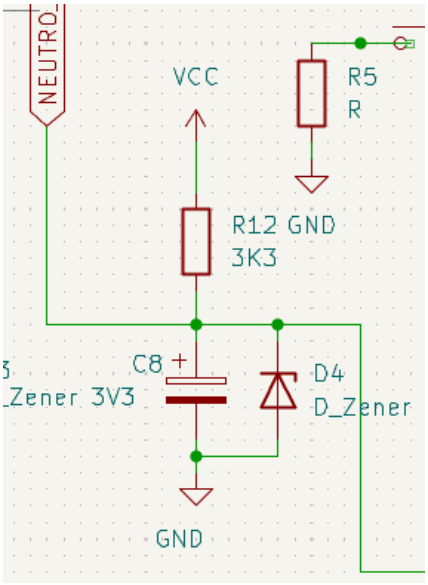
No teste embarcado, foi realizada a ligação da placa no chicote original da motocicleta, utilizando os cabos 12V que acionam as lâmpadas originais como entradas.

Para acionamento da luz indicadora de neutro, o acionamento é realizado com GND, mantendo o 12V ligado. O circuito inicial da placa é alterado para que funcione corretamente nessa condição.

As luzes de iluminação do painel apenas acionam quando o motor é ligado, impossibilitando a ligação do painel quando a chave de ignição é acionada. Para alimentação foi utilizado o fio preto do chicote, que funciona com 12V direto da bateria e acionado no pós-chave, evitando modificações no sistema da motocicleta.

A resistência da boia de combustível varia entre 10 Ω e 100 Ω, onde 10 Ω o sistema apresenta alto nível de combustível, e 100 Ω para tanque vazio. O código foi adaptado e o esquema foi alterado conforme a figura 62.

Figura 62 - Circuito Neutro Corrigido



Fonte: Autor (2025)

A figura 63 mostra superficialmente a conexão da placa na motocicleta em meio aos testes.

Figura 63 - Teste embarcado na motocicleta

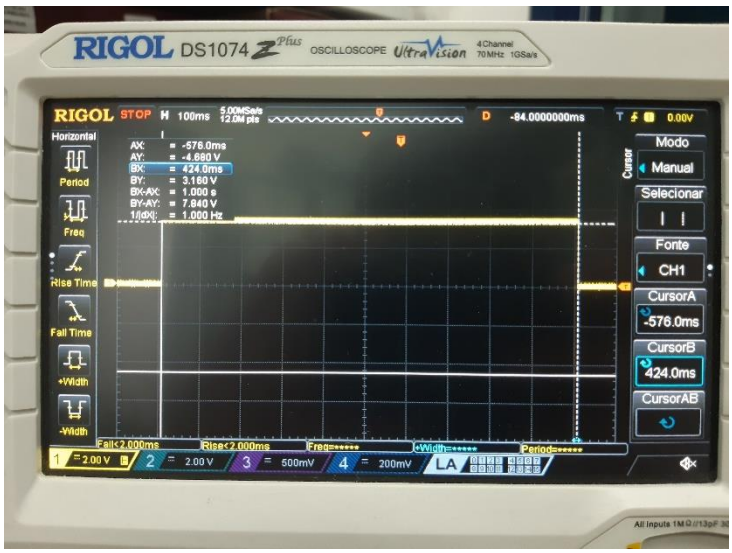
Fonte: Autor (2025)

4.1 Validação do tempo das tarefas do FreeRTOS

Para validação do tempo em que as tarefas executam foi realizado as medições por meio do osciloscópio, a fim de constatar se não há atrasos significativos nas execuções das tarefas. As medições apresentam uma margem de 1 milissegundo de erro.

A tarefa do nível de combustível realiza uma nova verificação a cada 1000 milissegundos configurados no FreeRTOS com o comando vTaskDelay(pdMS_TO_TICKS(1000));. É contatado a funcionalidade na figura 64.

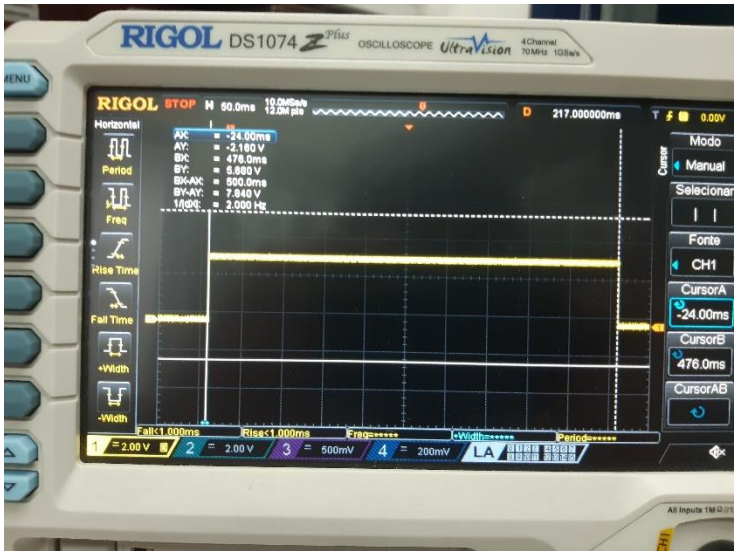
Figura 64 - Análise no osciloscópio tarefa combustível



Fonte: Autor (2025)

A tarefa do velocímetro não possui tempo de execução, porém é ativada pela tarefa de captação das bordas, que executa de 500 em 500 milissegundos. A figura 65 mostra a cadência da chamada da tarefa do velocímetro.

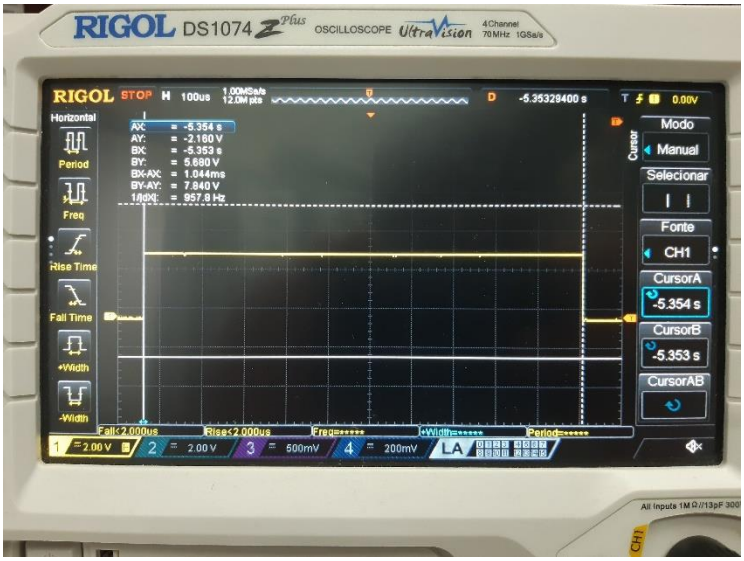
Figura 65 - Análise no osciloscópio tarefa velocímetro



Fonte: Autor (2025)

Nos mostradores, assim como o velocímetro a tarefa não possui tempo de repetição da execução, e é chamada pela interrupção causada pela troca de nível lógico na entrada dos GPIOs, logo a tarefa pode ser reexecutada em tempos menores, como visto na figura 66.

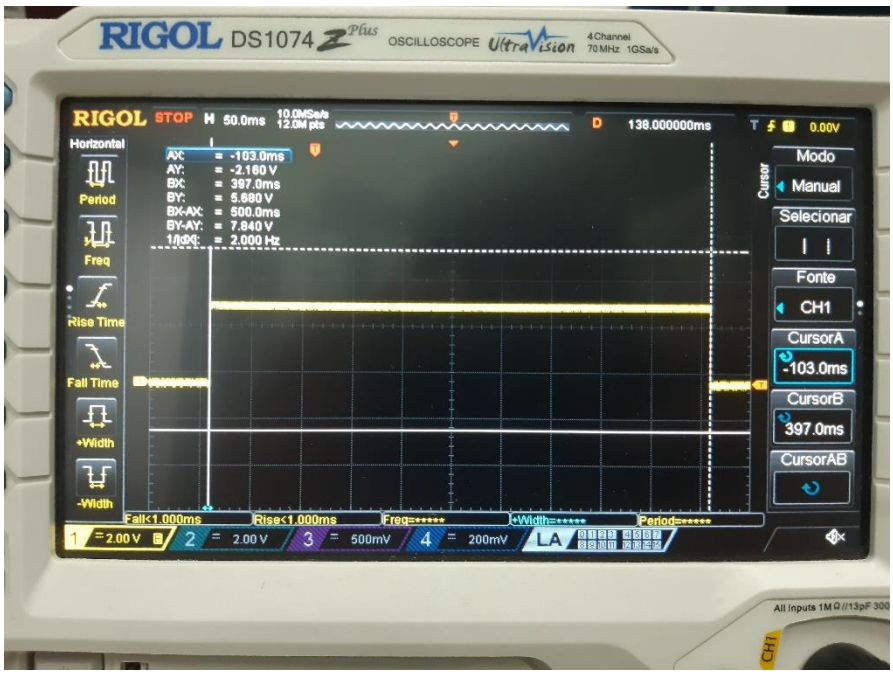
Figura 66 - Análise no osciloscópio tarefa mostradores



Fonte: Autor (2025)

Na tarefa de hodômetros a execução é cadenciada de 500 em 500 milissegundos, e pode ser averiguada na figura 67.

Figura 67 - Análise no osciloscópio tarefa hodômetro



Fonte: Autor (2025)

4.2 Resultados

Após a montagem do sistema e sua integração à motocicleta, foi possível validar o funcionamento do Cluster Digital com base na leitura dos sensores e na exibição das informações em tempo real na tela TFT. O microcontrolador Raspberry Pi Pico RP2040 foi capaz de processar os sinais provenientes dos sensores de velocidade, tensão e outros parâmetros, apresentando-os graficamente por meio da biblioteca LVGL. A montagem do Cluster na motocicleta é vista na figura 68.

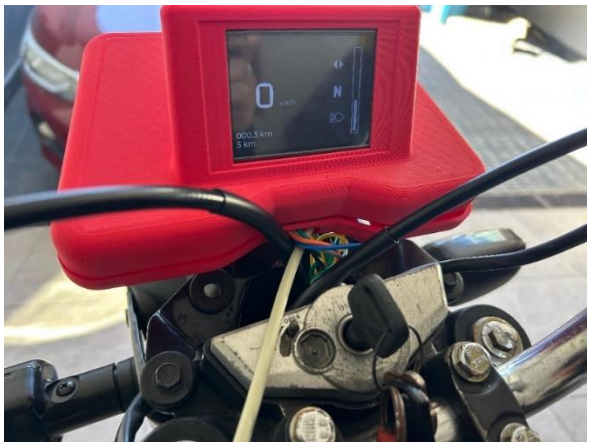
Figura 68 - Cluster montado na motocicleta

Fonte: Autor (2025)

Os testes demonstraram que a taxa de atualização do *display* foi satisfatória, garantindo uma visualização fluida dos dados, sem atrasos perceptíveis. No entanto, por não ser um display dedicado a aplicação automotiva apresentou baixa luminosidade, mesmo em brilho máximo. O sensor de campo magnético, acoplado ao cabo do velocímetro mecânico, mostrou-se eficaz na conversão do movimento rotacional em sinal digital, permitindo o cálculo preciso da velocidade, apesar de ser

um ponto importante de melhoria. A visualização mais próxima ao *display* do Cluster pode ser vista na figura 69.

Figura 69 - Cluster ligado na motocicleta



Fonte: Autor (2025)

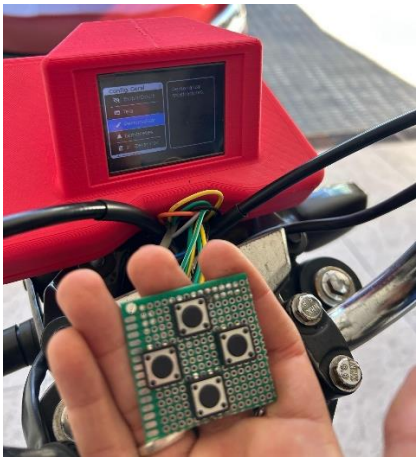
A *interface* gráfica desenvolvida apresentou boa legibilidade e organização das informações. Além disso, o sistema *plug-and-play*, proposto como um dos requisitos do projeto, funcionou conforme o esperado, utilizando o chicote original da motocicleta, conforme figura 70.

Figura 70 - Ligação Cluster na motocicleta

Fonte: Autor (2025)

A navegação do menu com o sistema montado na motocicleta foi possível, apesar de não possuir a base plástica que acopla o controle no guidão da motocicleta. Na figura 71 pode-se ver a navegação do menu.

Figura 71 - Menu de configurações montado na motocicleta



Fonte: Autor (2025)

Para análise dos circuitos foram realizados testes individuais, gerando o resultado da tabela 3.

Tabela 3 - Testes na motocicleta

SISTEMA	TESTE	TEVE ÊXITO?	OBSERVAÇÕES	CORREÇÃO
NEUTRO	ENGATAR NEUTRO	SIM	ACIONADO COM 0V	NÃO NECESSÁRIA
FAROL ALTO	ACENDER FAROL ALTO	SIM, COM DETALHES	OSCILAÇÕES EM MARCHA LENTA	MUDANÇA NO SISTEMA DE CAPTAÇÃO
SETA DIREITA	LIGAR SETA PARA DIREITA	SIM	FUNCIONAMENTO ADEQUADO	NÃO NECESSÁRIA
SETA ESQUERDA	LIGAR SETA PARA ESQUERDA	SIM	FUNCIONAMENTO ADEQUADO	NÃO NECESSÁRIA
COMBUSTIVEL	COMPARAR VALOR COM PAINEL ORIGINAL	SIM	VARIAÇÕES NO NÍVEL	MAIOR MÉDIA MÓVEL
VELOCIMETRO	GIRAR A RODA	SIM	BAIXA VELOCIDADE	APLICAÇAO DO SUPORTE PARA O CABO
HODOMETROS	CALCULAR DISTÂNCIA	SIM	MARGEM DE ERRO	APRIMORAR FÓRMULA DE CÁLCULO

Fonte: Autor (2025)

5 CONCLUSÃO

Este trabalho teve como objetivo o desenvolvimento de um Cluster Digital compatível com a motocicleta Honda CG 150 (ano 2007), sem que fossem necessárias modificações nos sistemas originais do veículo. O projeto buscou aliar funcionalidades modernas a um aspecto tecnológico e atual, promovendo uma atualização significativa em relação ao painel analógico original.

Ao longo do desenvolvimento, foram realizados estudos abrangentes sobre a evolução dos painéis de instrumentos, desde suas origens até as tendências atuais e futuras do mercado automotivo. Essa base teórica foi fundamental para direcionar corretamente as decisões de projeto e possibilitar a integração prática dos conhecimentos adquiridos durante o curso de Eletrônica Automotiva na FATEC.

Durante a execução, novas ideias e abordagens foram incorporadas, como o uso de tecnologias e ferramentas como LVGL, FreeRTOS e FreeCAD, além do aprofundamento em sistemas elétricos de motocicletas. Essa ampliação de conhecimentos foi essencial para transformar conceitos teóricos em um protótipo funcional, demonstrando a viabilidade prática do projeto e sua aplicabilidade no contexto real.

O projeto também contribui para a formação de novos profissionais da área, ao servir como base técnica e inspiradora para trabalhos futuros. Apesar das limitações impostas para adequação ao escopo institucional, o trabalho abre caminho para diversas melhorias e expansões.

Apesar dos bons resultados, observou-se que, para aplicações práticas, são desejáveis telas maiores e com maior capacidade de luminosidade, o que exigiria mais desempenho do microcontrolador. Isso aponta para a necessidade de futuras melhorias no projeto, como o uso de um *hardware* mais robusto, incluindo placas dedicadas (PCI) e sensores específicos para aplicações automotivas, assim como compatibilidade com sistemas de comunicação como ethernet e CAN, levando a ampliação da compatibilidade com outros modelos de motocicletas, maior reconhecimento de sinais de entrada, adoção de processadores mais robustos e o aumento da conectividade, aspecto que se mostra cada vez mais essencial na evolução dos *clusters* automotivos.

6 REFERÊNCIAS (GERAL)

CUNHA, Igor Bezerra Da. Controle Dos Acessórios Elétricos Do Veículo Por Tela Touch Screen. 2023. Disponível em: https://fatecsantoandre.edu.br/upload/66301ee51a197.pdf. Acesso em: 23 maio 2025.

CAPELLI, Alexandre. Eletroeletrônica Automotiva. São Paulo: Editora Érica, 2010. 368 p.

CAVALER, Hermes Júnior de Césaro. O Painel Automotivo – Como O Redesign Desta Peça Pode Ajudar Na Condução E Visibilidade Externa. 2019. Disponível em: http://repositorio.unesc.net/bitstream/1/9925/3/Hermes%20Júnior%20de%20Césaro%20Cavaler.pdf. Acesso em: 23 maio 2025.

CONTINENTAL. BLUME, Heinrich-Jochen; HAAS, Hermann. 111 Years of Speedometers: The History and Future of Driver Information. Ratisbona: PRC Werbe-GmbH, 2013. 111 p.

FREERTOS. What is FreeRTOS. Disponível em: https://www.freertos.org/Why-FreeRTOS/What-is-FreeRTOS. Acesso em: 29 out. 2024.

GUIMARÃES, Alexandre de Almeida. Eletrônica Embarcada Automotiva. São Paulo: Editora Érica, 2007. 321 p.

HONDA. Diagramas elétricos – Titan 150. 2007. Disponível em: https://pt.scribd.com/document/748133255/DIAGRAMAS-ELETRICOS-TITAN-150. Acesso em: 27 maio 2025.

LVGL. LVGL Documentation. Disponível em: https://docs.lvgl.io/master/. Acesso em: 29 out. 2024.

MEDEIROS, Weslley Torres. Comunicação Serial Parte - I. Slide de aula apresentado na disciplina Tópicos Avançados de Comunicação, Eletrônica Automotiva – FATEC Santo André, Santo André, 2024.

FERREIRA, Arthur William Lourenço. Análise De Desempenho Em Cluster Veicular Utilizando Teoria De Filas. Marabá, 2016. Disponível em: https://repositorio.unifesspa.edu.br/bitstream/123456789/227/1/TCC_Análise%20de

%20desempenho%20em%20cluster%20veicular%20utilizando%20teoria%20de%20Filas.pdf. Acesso em: 23 maio 2025.

PIRELLI. O painel: uma evolução contínua da informação – Desde um simples auxílio à condução até um conjunto de controles e eletrônicos. Disponível em: https://www.pirelli.com/global/pt-br/road/carros/o-painel-uma-evolucao-conti-nua-da-informacao-127781/. Acesso em: 09 nov. 2024.

RASPBERRY PI FOUNDATION. Raspberry Pi Pico Documentation. Disponível em: https://www.raspberrypi.com/documentation/microcontrollers/pico-series.html. Acesso em: 29 out. 2024.

SANTOS JÚNIOR, Álvaro Santana dos. Sistema De Monitoramento Eletrônico Automotivo. 2009. Trabalho de Conclusão de Curso (Graduação em Engenharia de Computação) – Centro Universitário de Brasília (UniCEUB), Brasília, 2009. Disponível em: https://repositorio.uniceub.br/jspui/bitstream/123456789/3211/2/20515930.pdf. Acesso em: 23 maio 2025.

TEIXEIRA, Antonio Carlos. Marcador de combustível: funcionamento, dúvidas, cuidados e prevenção para evitar a pane seca. Instituto Combustível Legal. Disponível em: https://institutocombustivellegal.org.br/marcador-de-combustivel-funcionamento-duvidas-cuidados-e-prevencao-para-evitar-a-pane-seca/. Acesso em: 28 maio 2025.

TEIXEIRA, João Pedro Alves. Teste e validação de um painel digital automóvel usando visão computacional. 2018. Dissertação (Mestrado Integrado em Engenharia Eletrotécnica e de Computadores) – Faculdade de Engenharia da Universidade do Porto, Porto, 2018. Disponível em: https://repositorio-aberto.up.pt/bitstream/10216/114105/2/277842.pdf. Acesso em: 23 maio 2025.

THOMAZINI, Daniel; ALBUQUERQUE, **Pedro Urbano Braga**. Sensores Industriais: fundamentos e aplicações. 4. ed. São Paulo: Érica, 2006. 372 p.

UNITED NATIONS ECONOMIC COMMISSION FOR EUROPE (UNECE). Addendum 38 – Regulation No. 39: Uniform provisions concerning the approval of vehicles with regard to the speedometer and odometer equipment including its installation. Genebra: United Nations, 2018.

6.1 REFERÊNCIAS DE IMAGENS

ACHARYA, Prakash. Unraveling Auto Tech: ECU Supply Chain Challenges and Innovations. LinkedIn, 14 ago. 2023. Disponível em: https://www.linkedin.com/pulse/unraveling-auto-tech-ecu-supply-chain-challenges-prakash-acharya/. Acesso em: 28 maio 2025.

AUTOPAPO. Luzes do painel do carro: saiba identificar o problema. Disponível em: https://autopapo.com.br/noticia/luzes-do-painel-do-carro-significado-problema/. Acesso em: 11 out. 2024.

BBC. Carruagem do rei Charles na coroação terá ar-condicionado e vidros elétricos. Disponível em: https://www.bbc.com/portuguese/articles/c72qyzr39d2o. Acesso em: 05 out. 2024.

CASTRO, Bernardo. Conta-giros: entenda como ele pode te ajudar a economizar combustível. AutoPapo, 22 maio 2023. Disponível em: https://autopapo.com.br/noticia/conta-giros/. Acesso em: 28 maio 2025.

CHAVELO CHAVES E VELOCÍMETROS LTDA. Velocímetros. Disponível em: https://www.chavelo.com.br/veloc%C3%ADmetros. Acesso em: 28 maio 2025.

CHEVROLET. Manual do proprietário – Painel de instrumentos. Disponível em: https://meu.chevrolet.com.br/content/dam/gmownercenter/gmsa/gmbr/dynamic/manuals/2023/chevrolet/Onix/pt/om_ng-chevrolet_Onix_my23-pt_BR.pdf. Acesso em: 09 out. 2024.

CNN. Significado das luzes do painel: entenda o que seu carro quer te contar. Disponível em: https://www.cnnbrasil.com.br/auto/luzes-do-painel/. Acesso em: 10 nov. 2024.

DE VOLTA PARA O VINIL. Rádio Valvulado SEMP AC-431 - 1951. Disponível em: https://www.devoltaparaovinil.com.br/2013/09/radio-valvulado-semp-ac-431-1951.html. Acesso em: 13 out. 2024.

ESTADÃO. Mercedes-Benz Classe E 2023 tem câmera para Tiktok e reunião no Zoom. Disponível em: https://jornaldocarro.estadao.com.br/carros/mercedes-benz-classe-e-2023-tem-camera-para-tiktok-e-reuniao-no-zoom/. Acesso em: 19 out. 2024.

HONDA. Manual do proprietário. Disponível em: https://www.honda.com.br/pos-venda/motos/sites/customer_service_motos/files/manuais/CG%20150%20Titan,%20Sport%20e%20Job%202007.pdf. Acesso em: 12 out. 2024.

INSTITUTO NEWTON C. BRAGA. As Válvulas - O que você precisa saber sobre esses componentes antigos! (V001). Disponível em: https://www.newtoncbraga.com.br/mundo-das-valvulas/457-as-valvulas-o-que-voce-precisa-sobre-esses-componentes-antigos-v001.html?showall=1. Acesso em: 10 nov. 2024.

INSTITUTO NEWTON C. BRAGA. Como funciona o velocímetro (ART1454). Disponível em: https://www.newtoncbraga.com.br/como-funciona/8403-como-funciona-o-velocimetro-art1454.html. Acesso em: 15 out. 2024.

JVIS USA, LLC. Integrated Center Stacks. Disponível em: https://jvis.us/product/integrated-center-stacks/. Acesso em: 28 maio 2025.

MOTOSBLOG. Ficha técnica da Honda CG 150 Titan KS 2004 a 2009. Disponível em: https://www.motonline.com.br/fipe/honda/cg-150-titan-es/cg-150-titan-es/2007. Acesso em: 11 out. 2024.

QUATRO RODAS. Mercury Eight: símbolo de rebeldia. Disponível em: https://quatrorodas.abril.com.br/noticias/mercury-eight-simbolo-de-rebeldia. Acesso em: 18 out. 2024.

QUATRO RODAS. Miura Saga: olha quem está falando! Disponível em: https://quatrorodas.abril.com.br/noticias/miura-saga-olha-quem-esta-falando. Acesso em: 18 out. 2024.

QUATRO RODAS. Quando andamos com o carro de ré, isso é contado no hodômetro? Disponível em: https://quatrorodas.abril.com.br/auto-servico/quando-andamos-com-o-carro-de-re-isso-e-contado-no-hodometro. Acesso em: 11 out. 2024.

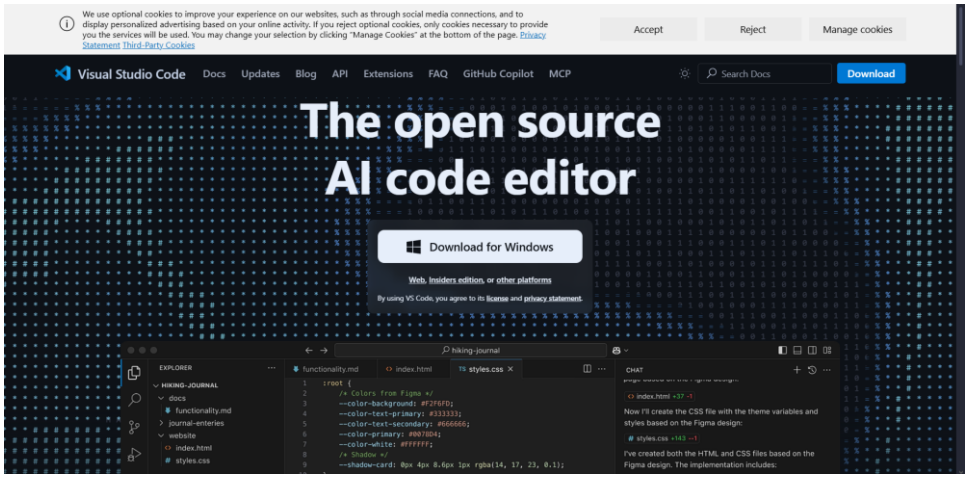
RACTRONICOS. A história do painel de instrumentos: da placa de madeira à projeção em 3D. Disponível em: https://www.ractronicos.pt/noticias/a-historia-do-painel-de-instrumentos-da-placa-de-madeira-a-projecao-em-3d. Acesso em: 11 out. 2024.

WHEELSAGE. Painel Opel Kadett GSi 3-door (E) '1984–91. Disponível em: https://br.wheelsage.org/opel/kadett/pictures/jpacth. Acesso em: 18 out. 2024.

7 APÊNDICE

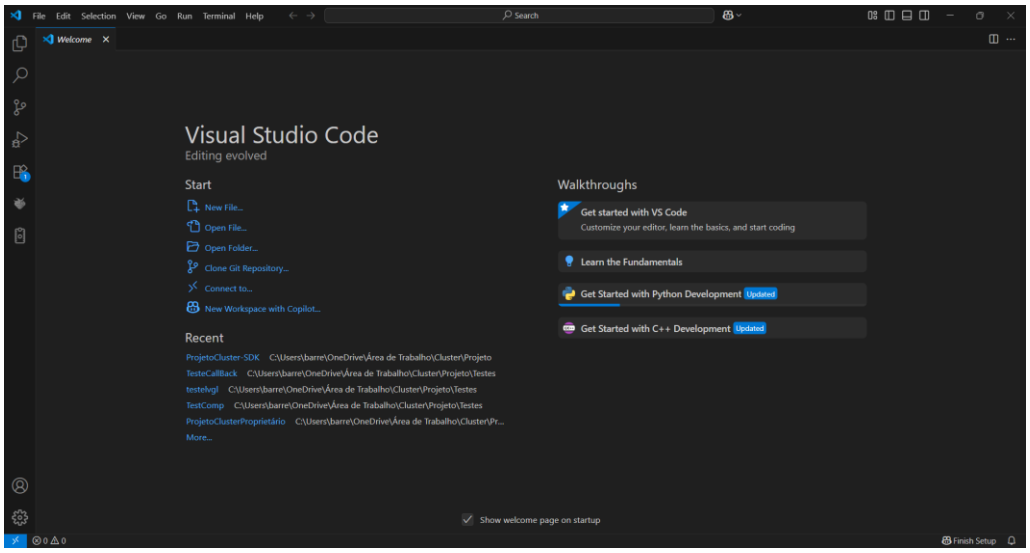
7.1 Obtenção do VSCode

Para iniciar o processo de programação deve-se obter o VSCode. O link para download da ferramenta da Microsoft se encontra em https://code.visualstudio.com.

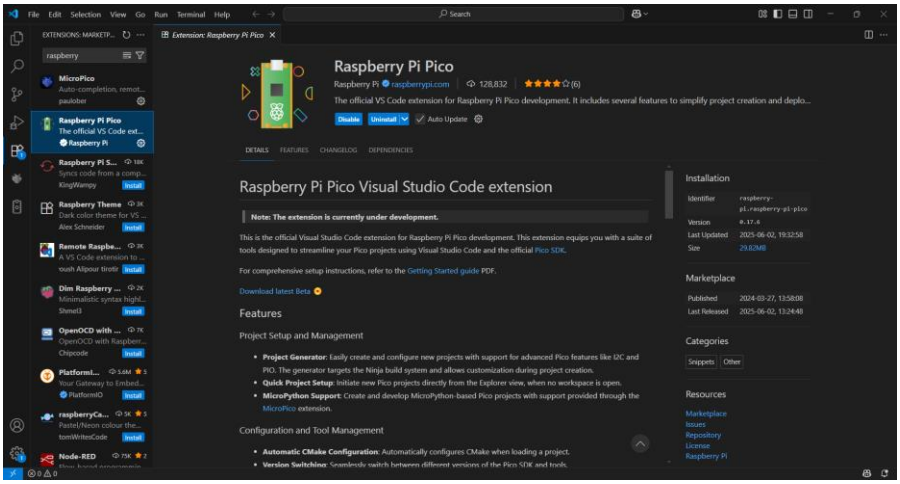


Selecionamos a opção de *download* e o sistema operacional do computador para início do *download*. Com o arquivo baixado, realizamos a execução e instalação da aplicação na máquina, seguindo o passo a passo fornecido pelo assistente de instalação.

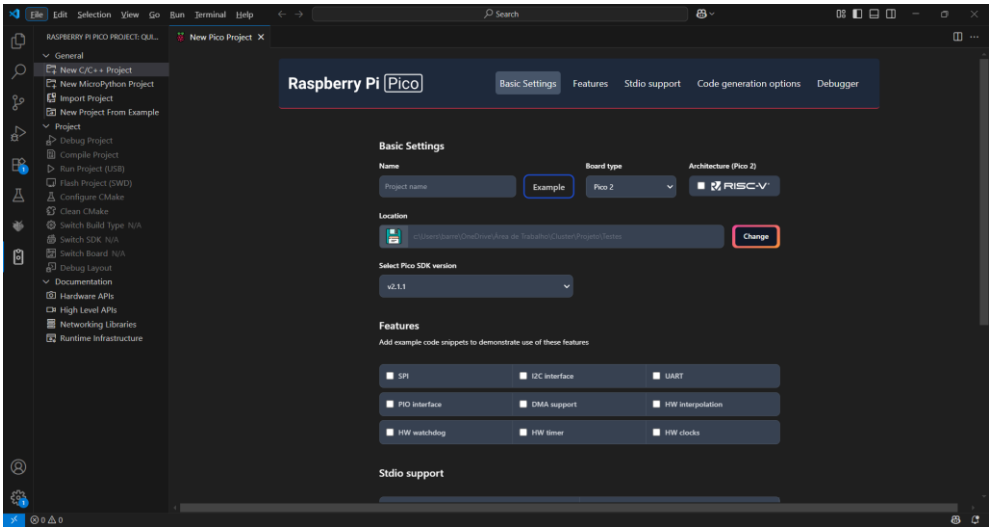
Com a ferramenta devidamente instalada, o programa é aberto e é realizada a adição das extensões para uso do Raspberry.



Na barra lateral direita há o ícone “extensões”, pesquisamos “Raspberry PI Pico” na barra de pesquisa e localizamos a extensão da Raspberry. Utilizamos a opção de “instalar” para adição da extensão no VSCode. A extensão automaticamente já adiciona outras extensões também necessárias como CMake.



Após a adição das extensões, pode-se criar um novo projeto. Abre-se a extensão do Raspberry e seleciona-se a opção “Novo Projeto C/C++”. Aqui configura-se o nome, tipo de placa, local do projeto, versão da SDK, além de habilitar recursos do microcontrolador.



O primeiro projeto a ser criado realizará o *download* da SDK, esse processo ocorrerá cada vez que for obtido uma nova versão de SDK que não esteja presente no computador. Baixar novas SDKs pode levar algum tempo adicional.

Após concluído a criação do projeto basta iniciar a programação. Caso seja necessário um teste de validação, pode-se utilizar a opção “Novo projeto de exemplo” onde já possui linhas de códigos prontas para teste de terminadas funções, como de piscar o *LED* da placa no exemplo “*blink*”. Com o projeto de exemplo carregado, abrirá uma barra inferior com as opções de compilação e outras ferramentas do VSCode, utilizamos a opção “compilar” para a criação do arquivo “.uf2” com o hexadecimal reconhecido pela máquina. Para programas maiores a primeira compilação pode demorar alguns minutos.

7.2 Gravação na placa

Para realizar a gravação na placa é necessário que o microcontrolador seja conectado ao computador com o botão “*bootsel*” pressionado, o dispositivo se conecta como um *pendrive*.

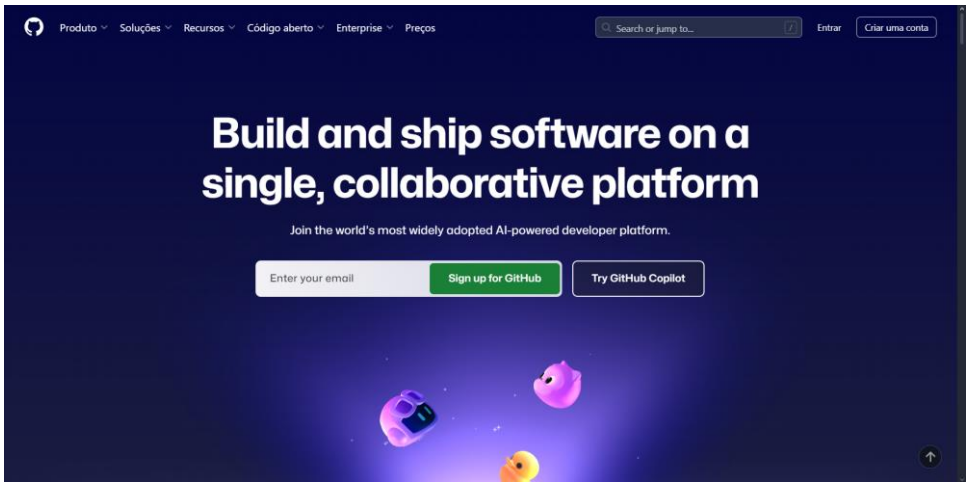
O arquivo gerado pelo compilador se localiza na pasta raiz onde o projeto foi criado. Dentro da pasta raiz podemos localizar a subpasta “*build*”, onde conterá com arquivo “.uf2”, antecedido pelo nome dado ao projeto. Realiza-se a cópia do arquivo que será enviada para a unidade do microcontrolador. O arquivo e copiado e logo após a PI Pico se reiniciará dando início a aplicação.

7.3 GitHub

Conforme citado anteriormente, é necessário o uso do GitHub para adição das bibliotecas auxiliares e versionamento em nuvem do código. O *download* e instalação é pelo link https://git-scm.com/downloads, ou uso diretamente pelo terminal interno do VSCode, onde não necessitará de instalação adicional. Pode-se obter também a Git GUI, onde há uma interface que elimina o uso do terminal, realizando a clonagem dos repositórios apenas através do *link*.

Por padrão, o VSCode adiciona a extensão do GitHub, acessado por meio da opção de “*Source Control*” na barra lateral direita. Por meio dessa extensão será realizado os *commits*, as versões do código, e sincronismo com a nuvem.

Para utilizar o GitHub é necessário que o usuário possua uma conta Git, criada pelo site https://github.com. Apenas para clonagem dos repositórios não é necessário que uma conta esteja vinculada, apenas para versionamento em nuvem.



7.4 Pico-displayDrivs

Para a comunicação com o display ILI9341 utiliza-se uma biblioteca capaz de enviar as imagens e que serão integradas ao LVGL posteriormente. O *link* da biblioteca se encontra em https://github.com/tvlad1234/pico-displayDrivs.git. Localiza-se a pasta raiz do projeto utilizando o comando “cd” através do prompt de comando CMD do Windows, “cd patch”, já localizando a pasta “lib” criada para armazenar todas as bibliotecas. O comando de clonagem é “git submodule add <URL_do_repositório>”, especificamente “git submodule add https://github.com/tvlad1234/pico-displayDrivs.git” realizando o clone do repositório como submódulo.

Para vincular a biblioteca ao projeto precisamos realizar a configuração através do arquivo do CMake. No CMake da pasta “lib” é adicionado as linhas “add_subdirectory(pico-displayDrivs)”. Como a biblioteca também conta com compatibilidade para outros displays, pode-se comentar a adição de bibliotecas que não serão utilizadas afim de economia de tempo na compilação e memória. Na pasta “ili9341” há os arquivos “.c” e “.h” que possuem a declaração das funções e outras definições. No arquivo “.c“ pode-se configurar quais pinos serão utilizados para comunicação SPI. Por padrão utiliza-se as portas de spi0 para comunicação, pinos 16 a 20. No programa principal realiza-se a chamada da função LCD_initDisplay(); para que a comunicação funcione. Caso seja necessário teste da funcionalidade a biblioteca conta com a subpasta gfx, uma biblioteca gráfica capaz de gerar os gráficos para serem exibidas no display. Vide o arquivo README.md para maiores informações.

7.5 LVGL

Similar ao processo de clonagem da biblioteca pico-displayDrivs, realiza-se a clonagem da biblioteca do LVGL por meio do *link* https://github.com/lvgl/lvgl.git, com o comando “git submodule add https://github.com/lvgl/lvgl.git”. A versão do LVGL utilizada no projeto foi a 9.2 as demais instruções foram baseadas na experiência do autor nesta versão do LVGL. Com o repositório clonado adiciona-se ao CMake a linha “add_subdirectory(lvgl)”. Para que o LVGL funcione é necessário que um arquivo “lv_config.h” seja adicionado com as configurações necessárias. Um *template* é disponibilizado com a explicação de cada configuração, deve-se adequar a placa utilizada. A memória deve ser racionada para que seja utilizada pelo FreeRTOS e LVGL, já que utilizam um sistema de memória predefinido por configuração e não por memória dinâmica. Para o projeto foi utilizado 128 quilobytes dos 256 quilobytes disponíveis para o LVGL, dado pela configuração “#define LV_MEM_SIZE (128 * 1024U)”, já que o LVGL consome maior volume de memória por processamento dos gráficos.

O LVGL também apresenta funções que devem ser chamadas no início do programa, a função “lv_init();”, e include do arquivo header do LVGL por meio do “lvgl.h”.

Outros parâmetros devem ser definidos conforme requisição do LVGL. Um display “virtual” para que o LVGL se adeque ao tamanho da tela utilizada e gerar a imagem de saída adequada:

```
lv_display_t *display = lv_display_create(MY_DISP_HOR_RES, MY_DISP_VER_RES);
```

Onde o display é criado, passando as resoluções horizontais e verticais do display utilizado.

```
lv_display_set_rotation(display, LV_DISPLAY_ROTATION_90);
```

Definindo a orientação que a tela será utilizada, note que a configuração se da pela mesma variável display criada anteriormente.

```
lv_display_set_buffers(display, buf1, buf2, sizeof(buf1), LV_DISPLAY_RENDER_MODE_PARTIAL);   /*Initialize the display buffer.*/
```

Essa função define os *buffers* que serão utilizados pelo LVGL para guardar as informações para atualizar o display.

O tipo de cor utilizada pelo display é configurado. No caso do ILI9341, utiliza-se RGB565, através desta informação calcula-se o a densidade dos pixels.

```
#define BYTE_PER_PIXEL (LV_COLOR_FORMAT_GET_SIZE(LV_COLOR_FORMAT_RGB565))
```

É passado o display criado, variáveis de buffer, nesse caso utilizado dois buffers, definidos conforme o exemplo do LVGL:

```
static uint8_t buf1[MY_DISP_HOR_RES * MY_DISP_VER_RES / 10 * BYTE_PER_PIXEL];

static uint8_t buf2[MY_DISP_HOR_RES * MY_DISP_VER_RES / 10 * BYTE_PER_PIXEL];
```

O modo de renderização é modo parcial, onde o LVGL atualiza a tela por meio de pequenas partes renderizadas, modo mais leve.

Com todas as configurações do LVGL configuradas é necessário que seja feita a integração com a biblioteca que comunica com o display. A função de saída do LVGL deve ser criada pelo usuário, e conforme exemplo é dada por:

```
lv_display_set_flush_cb(display, my_flush_cb_2);
```

Onde é definido qual a função de atualização.

```
void my_flush_cb_2(lv_display_t * disp, const lv_area_t * area, uint8_t * px_map)
{
    uint16_t size_y = (area->y2 - area->y1) + 1;
    uint16_t size_x = (area->x2 - area->x1) + 1;
    LCD_WriteBitmap(area->x1, area->y1, size_x,size_y, (uint16_t *)px_map);
    lv_display_flush_ready(disp);
```

}

Esta função recebe qual display foi configurado, o tamanho da área a ser atualizada, uma vez que a tela não atualiza completamente, e o “px_map” que possui as cores.

A função está otimizada e adaptada, convertendo as posições de estrutura de lv_area para uint16_t e passando as informações por meio de bitmap pela função da biblioteca pico-displayDrivs. Quando o display é atualizado é avisado ao LVGL pela função lv_display_flush_ready();.

O LVGL está pronto para uso.

7.6 FreeRTOS

A biblioteca que gerenciará o programa por meio de tarefas e controle de tempo é o FreeRTOS. É realizada a adição do submódulo pelo link https://github.com/FreeRTOS/FreeRTOS-Kernel.git, linha de comando “git submodule add https://github.com/FreeRTOS/FreeRTOS-Kernel.git”, com a pasta “lib” aberta. Por se tratar de uma biblioteca que gerenciará diretamente o uso do processamento da Raspberry Pi Pico, esta biblioteca apresenta alguns detalhes a serem adicionados. Os arquivos se iniciam pelo arquivo de “import.cmake”, no arquivo principal do CMake.

```
include(${FREERTOS_SRC_DIRECTORY}/portable/ThirdParty/GCC/RP2040/FreeRTOS_Kernel_import.cmake)
```

Onde “FREERTOS_SRC_DIRECTORY” é o caminho de pastas a partir do projeto de onde está localizada a pasta do FreeRTOS-Kernel.

Ainda no arquivo principal do CMake, adicionamos os arquivos como “libraries” e diretórios, configurando a “heap” e “port.c” compatíveis com o microcontrolador.

```
add_library(FreeRTOS INTERFACE
    ${FREERTOS_SRC_DIRECTORY}/event_groups.c
    ${FREERTOS_SRC_DIRECTORY}/list.c
    ${FREERTOS_SRC_DIRECTORY}/queue.c
    ${FREERTOS_SRC_DIRECTORY}/stream_buffer.c
    ${FREERTOS_SRC_DIRECTORY}/tasks.c
```

```
    ${FREERTOS_SRC_DIRECTORY}/timers.c
    ${FREERTOS_SRC_DIRECTORY}/portable/MemMang/heap_3.c
    ${FREERTOS_SRC_DIRECTORY}/portable/GCC/ARM_CM0/port.c
)
```

Inclui-se outros diretórios, adicionando o arquivo de arquitetura compatível com o processador do microcontrolador:

```
target_include_directories(FreeRTOS SYSTEM
INTERFACE
    .
    ${FREERTOS_SRC_DIRECTORY}/include
    ${FREERTOS_SRC_DIRECTORY}/portable/GCC/ARM_CM0
)
```

Com estas linhas está configurado o FreeRTOS pelo CMake. Assim como o LVGL, o FreeRTOS também apresenta um arquivo de configurações internas, chamado de “FreeRTOSConfig.h”, que também deve ser configurado conforme a placa utilizada. Um *template* com a explicação das configurações também é disponibilizado pela desenvolvedora para auxílio na configuração das funcionalidades.

No projeto principal, adiciona-se os *headers* do FreeRTOS com “#include "FreeRTOS.h"”, sendo este o principal além de outros arquivos que devem ser adicionados conforme uso das funcionalidades, como “task.h” (geralmente incluído em todos os casos), “semphr.h” e “event_groups.h”, utilizados neste projeto.

7.7 Integração LVGL e FreeRTOS

Para que o LVGL e FreeRTOS funcionem corretamente é necessário que sejam realizadas configurações no LVGL. As configurações reduzem o uso do processamento, memória RAM e possibilitam integração de funções. O LVGL apresenta compatibilidade com sistemas operacionais como FreeRTOS entre outros sistemas. Habilita-se o FreeRTOS no lv_conf.g pela configuração:

```
#define LV_USE_OS   LV_OS_FREERTOS
```

Esta etapa pode apresentar erros de integração por não reconhecimentos de outras pastas, para isso no arquivo CMakelists.txt da pasta principal do LGVL, foi adicionado as linhas:

```
target_include_directories(lvgl PUBLIC
   ${CMAKE_CURRENT_LIST_DIR}/../FreeRTOS-Kernel/include
   ${CMAKE_CURRENT_LIST_DIR}/../FreeRTOS-Kernel/portable/GCC/ARM_CM0
)
target_link_libraries(lvgl INTERFACE
   FreeRTOS
)
```

Ambas as bibliotecas funcionarão juntas, rodando o programa com gerenciamento do tempo e criação de imagens de *interface* com o usuário, sendo transmitidas ao display por meio da biblioteca pico-displayDrivs, que atua como um driver.

7.8 Método de programação

Para criação do programa foram utilizadas tarefas do FreeRTOS rodando a programação do LVGL. A tarefa é gerenciada pelo FreeRTOS, cada tarefa possui sua prioridade e em seu escopo são adicionados *delays* que não travam o processamento do microcontrolador, cadenciando a aplicação. Cada tarefa roda em seu início o código do LVGL para que sejam criados os objetos na tela. As tarefas possuem loops internos e são utilizados para que os objetos sejam atualizados conforme novos valores são obtidos, contendo um tempo fixo de atualização.

A integração entre tarefas é feita por meio de *event groups* e semáforos que liberam e sinalizam as tarefas, funcionam como mensageiros para que funções entre tarefas sejam possíveis.