UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA - CURSO DE ENGENHARIA MECÂNICA
TRABALHO DE CONCLUSÃO DE CURSO

Análise da influência do modo de condução em um veículo de passeio via a leitura da Rede CAN 2.0 com o uso de Arduino

por

Pedro Henrique Raznievski Silveira

Monografia apresentada ao Departamento de Engenharia Mecânica da Escola de Engenharia da Universidade Federal do Rio Grande do Sul, como parte dos requisitos para obtenção do diploma de Engenheiro Mecânico.

Porto Alegre, dezembro de 2019

CIP - Catalogação na Publicação

Silveira, Pedro Henrique Raznievski
Análise da influência do modo de condução em um veículo de passeio via a leitura da Rede CAN 2.0 com o uso de Arduino / Pedro Henrique Raznievski Silveira. -- 2019.
27 f.
Orientador: Alexandre Vagtinski De Paula.

Trabalho de conclusão de curso (Graduação) -- Universidade Federal do Rio Grande do Sul, Escola de Engenharia, Curso de Engenharia Mecânica, Porto Alegre, BR-RS, 2019.

1. Rede CAN. 2. Computador de bordo. 3. Arduino. 4. Modo de condução. 5. Veículos de passeio. I. De Paula, Alexandre Vagtinski, orient. II. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da UFRGS com os dados fornecidos pelo(a) autor(a).

Pedro Henrique Raznievski Silveira

Análise da influência do modo de condução em um veículo de passeio via a leitura da Rede CAN 2.0 com o uso de Arduino

ESTA MONOGRAFIA FOI JULGADA ADEQUADA COMO PARTE DOS REQUISITOS PARA A OBTENÇÃO DO TÍTULO DE
ENGENHEIRO MECÂNICO
APROVADA EM SUA FORMA FINAL PELA BANCA EXAMINADORA DO CURSO DE ENGENHARIA MECÂNICA

Prof. Mario Roland Sobczyk Sobrinho
Coordenador do Curso de Engenharia Mecânica

Área de Concentração: Energia e Fenômenos de Transporte/Processos de Fabricação/Mecânica dos Sólidos

Orientador: Prof. Alexandre Vagtinski de Paula

Comissão de Avaliação:

Prof. Cirilo Seppi Bresolin

Prof. Sérgio Luiz Frey

Prof. Paulo Smith Schneider

Porto Alegre, dezembro de 2019

AGRADECIMENTOS E DEDICATÓRIA

Gostaria de agradecer a todos que de maneira indireta ou direta contribuíram para minha formação pessoal, a qual foi imprescindível para ter êxito na conclusão desta etapa da vida. Entre os dedicados, gostaria de destacar a minha homenagem aos meus irmãos, Paulo Vinícius e Eduardo, e meus pais, Paulo Roberto e Rosane, pois eles são a minha fonte de apoio para conseguir superar todas as adversidades que a vida colocou em meu caminho. Por eles eu tenho motivo para acordar todo dia e buscar ser alguém melhor.

“O sucesso é ir de fracasso em fracasso sem perder entusiasmo.”

Winston Curchill

Silveira, Pedro Henrique Raznievski. **Análise da influência do modo de condução em veículos de passeio via a leitura da Rede CAN 2.0.2019 com o uso de Arduino**. Número de páginas: 20. Monografia de Trabalho de Conclusão do Curso em Engenharia Mecânica – Curso de Engenharia Mecânica, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2019.

RESUMO

Atualmente existe uma grande preocupação em busca da melhor utilização dos recursos energéticos por parte dos automóveis de passeio. Entretanto, estes automóveis não alcançam o melhor rendimento devido à maneira como são conduzidos, ocasionada em parte pelo desconhecimento dos motoristas sobre a melhor forma de operá-los. Constatado esse problema, o presente trabalho busca aferir qual é esta maneira mais eficiente de operá-los com base na execução de testes em um perímetro urbano junto da leitura de parâmetros fornecidos pela rede CAN 2.0 do automóvel via um computador de bordo desenvolvido. O computador de bordo desenvolvido foi programado para a leitura dos seguintes parâmetros: a carga calculada do motor, temperatura do líquido de arrefecimento, rotação do motor, velocidade do veículo, nível de combustível e o consumo de combustível. Executados os testes e registradas as medições de tais fatores pode-se averiguar a influência sofrida destes parâmetros conforme a maneira de condução, assim como responder a dúvida motivacional do trabalho sobre qual a melhor forma de trafegar com um veículo de passeio demandando o menor consumo de combustível e desgaste dos componentes.

PALAVRAS-CHAVE: Consumo de combustível, modo de condução, rede CAN 2.0, computador de bordo, veículos de passeio.

Silveira, Pedro Henrique Raznievski. **Analysis of the influence of driving mode on passenger cars by reading CAN bus 2.0 using Arduino**. 2019. Number of pages: 20. Mechanical Engineering End of Course Monography – Mechanical Engineering degree, The Federal University of Rio Grande do Sul, Porto Alegre, 2019.

ABSTRACT

Currently, there is a major concern for the better use of energy resources by passenger cars. However, these cars do not achieve the best performance because of the way they are driven, partly due to drivers' lack of knowledge about how to operate them. Noting this problem, the present work sought to assess what is the most efficient way to operate them based on the execution of tests in an urban perimeter by reading parameters provided by the car's CAN bus 2.0 using a developed onboard computer. This computer has been programmed to read the following parameters: calculated engine load, coolant temperature, engine speed, vehicle speed, fuel level and fuel consumption. After the tests are performed and the measurements of these factors are recorded, it is possible to ascertain the influence of these parameters according to the way of driving as well as to answer the motivational question about the best way to travel with a passenger vehicle, demanding the lowest fuel consumption and component attrition.

KEYWORDS: fuel consumption, driving mode, CAN bus 2.0, on board computer and passenger vehicles

SUMÁRIO

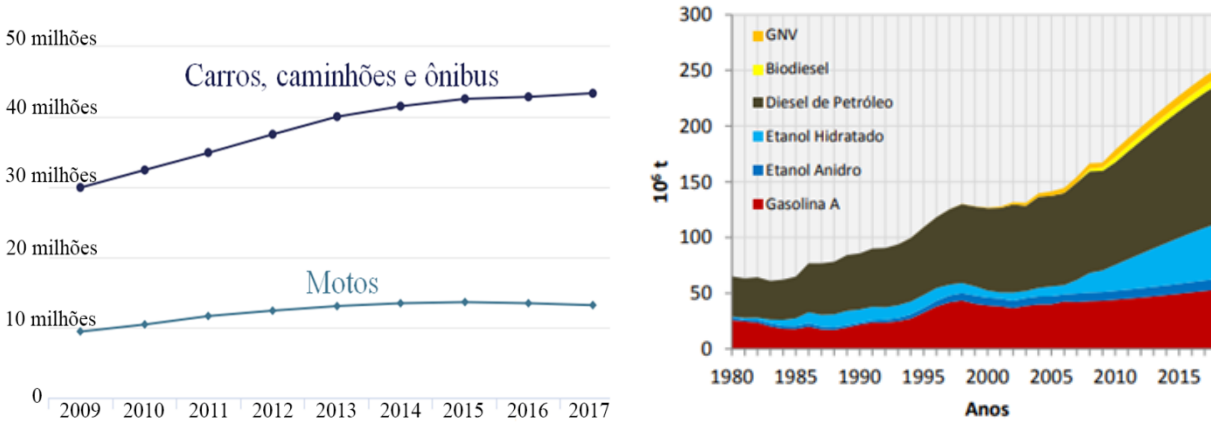
Pág.

1. INTRODUÇÃO

Atualmente, de acordo com o relatório da Sindipeças (2019), o Brasil apresenta uma frota circulante de cerca de 46 milhões de carros que contribuem para o crescimento das 250 milhões de toneladas de CO_2 produzidas por todas as classes de veículos, de acordo com o inventário do Ministério do Meio Ambiente (2011). A Figura 1 (a) apresenta o aumento da frota circulante de veículos no período de 2009 a 2017, enquanto a Figura 1 (b) ilustra o crescimento do volume de CO_2 emitidos ao longo dos últimos 40 anos pelo tipo de combustível.

Figura 1 – (a) Crescimento da frota circulante de veículos no Brasil; (b) A emissão de CO_2 por tipo de combustível ao longo dos últimos 40 anos.

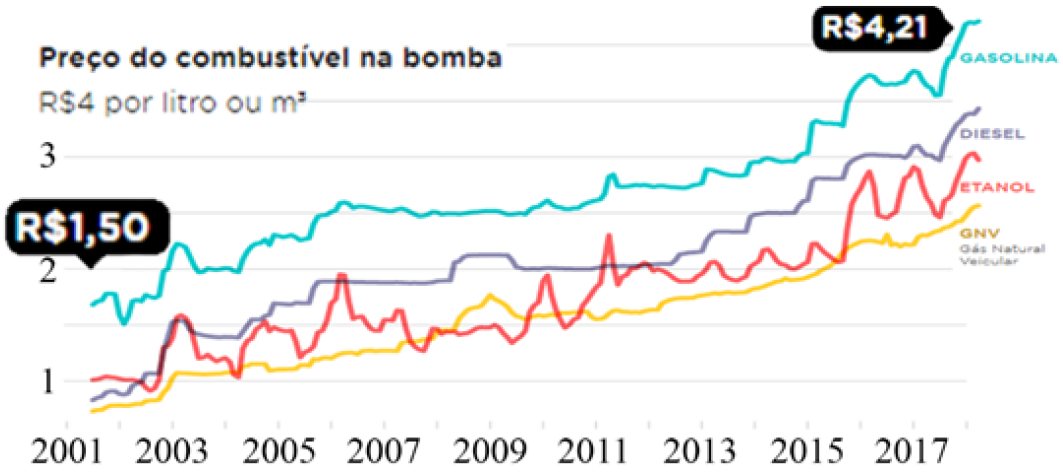
(a) (b)



Fonte: (a) Sindipeças; (b) Relatório final do 1° Inventário Nacional de emissões atmosféricas por veículos automotores rodoviários.

Esse elevado volume de poluentes acarreta **danos severos à sociedade** e tem sido motivo de muitos motoristas buscarem veículos com maior eficiência de combustível. Outro aspecto que intensifica essa procura por carros mais econômicos é o crescimento do preço dos combustíveis quase triplicarem nos últimos 18 anos de acordo com a ANP (2018). A Figura 2 apresenta a variação do preço do combustível ao longo dos anos de acordo com o tipo.

Figura 2 – Variação do preço dos combustíveis nos últimos 18 anos.



Fonte: Jornal Nexo.

Sendo assim, fica claro o grande movimento em buscar de uma maior eficiência no consumo dos combustíveis ao utilizar os automóveis. Consequentemente, o conhecimento adequado do funcionamento dos veículos, o que é possível via a leitura de um computador de bordo é de grande auxílio para se ter êxito nessa campanha.

2. MOTIVAÇÃO

Analisando os grandes problemas atuais da população, verificou-se a crise econômica e os autos indicies de poluição ocasionada pela emissão de um dos gases causadores do efeito estufa. Para combater esses dois problemas, decidiu-se desenvolver um computador de bordo de baixo custo para averiguar parâmetros de uso do veículo e buscar a maneira de trafegar com maior eficiência do combustível. Desta forma, motoristas de veículos populares conseguiriam economizar o dinheiro gasto com combustível, reduzir os gases poluentes emitidos pelo automóvel e, consequentemente ajudariam a enfrentar dois grandes problemas da população mundial.

3. REVISÃO BIBLIOGRÁFICA

Com o avanço da tecnologia nos veículos, cada vez se demanda um maior número de parâmetros que os motoristas devem controlar para se buscar a maneira de condução mais adequada e, consequentemente a melhor eficiência do combustível, maior vida útil dos componentes e a menor emissão de poluentes. Tendo conhecimento disso, as fabricantes têm desenvolvido os computadores de bordo que apresentam algumas informações importantes, as quais variam de acordo com a espécie do veículo.

Nas motocicletas do modelo Motorrad, a BMW (2016) informa que o computador de bordo de maneira rápida e sucinta apresenta as informações a fim de evitar causar uma distração no motociclista com um elevado número de parâmetros não essenciais. A fabricante declara que além dos parâmetros usuais (velocidade, consumo médio, nível de óleo e outros) de um computador de bordo, as motocicletas Motorrad apresentam a temperatura com aviso de formação de gelo, o qual é um possível risco em veículos que se deslocam em elevadas velocidades.

Já nos caminhões que trafegam em menor velocidade, com uma carga superior e emitem um volume maior de gases poluentes, os parâmetros necessários no computador de bordo são outros. Vide a fabricante Iveco (2019), que nos seus caminhões da linha Hi-Way apresenta ao condutor o nível de pressão nos freios, o tempo de funcionamento do motor e o nível de Arla 32.

Os automóveis, que serão o foco no presente trabalho, apresentam uma vasta configuração de computadores de bordo que varia de acordo com o padrão de luxo. Nota-se essa variação observando, por exemplo, uma GM Tracker, ano 2014, que segundo a Chevrolet (2013) apresenta em seu computador de bordo a velocidade média, o alcance com o combustível restante, consumo médio de combustível e o tempo de condução, enquanto um automóvel mais caro, como o Honda Civic, ano 2018, possui além de tais parâmetros, o consumo instantâneo, temperatura externa, GPS, vida útil do óleo, medidor de pressão do turbocompressor e outros, conforme informado pela montadora (Honda, 2018).

Entretanto, por mais luxuosos que sejam os automóveis, nem todas as informações são expostas, o que pode fazer falta na análise de um mecânico para diagnosticar algum problema.

Para sanar essa carência, a Bosch (2019) desenvolveu diferentes scanners que conectam na entrada OBD II (*On Board Diagnostics* - porta de acesso dos veículos) e permitem a leitura dos códigos genéricos de falhas e valores reais de diversos sensores. Dentre os modelos mais atuais dos leitores estão o modelo 1100 e o 1150. Na mesma mão, cita-se outra fabricante de scanners, a Raven (2019), que alega que seu produto, um Scanner de 3ª Geração tem a capacidade de executar as mesmas funções citadas pela concorrente e ainda a possibilidade de programar transmissões automatizadas, ativação de atuadores, ajustes eletrônicos, entre outras.

O grande problema dos scanners automotivos é o elevado preço, que dificulta um condutor não profissional de adquiri-los para aperfeiçoar sua maneira de condução e monitorar o estado do seu veículo. A opção para esses motoristas é a aquisição de scanners mais populares, como o Carrorama, da marca Multilaser (2019), a qual garante que o mesmo apresenta indicadores do veículo no celular do motorista via conexão Bluetooth, dentre os quais citam-se: a temperatura do líquido de arrefecimento, rotação de motor, tensão da bateria, mistura de etanol, chassi entre outros. O grande problema de alguns desses equipamentos populares são as opiniões dos usuários, que relatam imprecisão, atraso da informação e travamentos.

Visto essa falta de possibilidades aos motoristas, alguns estudantes de engenharia desenvolveram seus trabalhos de conclusão de graduação com o intuito de desenvolver um computador de bordo portátil.

O trabalho de Farinelli (2016) consistiu na leitura da rede CAN de um veículo via a utilização de um Arduino Uno e um Arduino Mega que, conectados ao um microcontrolador, modelo MCP2515, realizava a leitura de alguns parâmetros e apresentava em uma tela LCD. Já Almeida (2017) produziu um aplicativo no celular com sistema Android para avaliar o consumo de combustível. Estas informações eram enviadas via Bluetooth de um dispositivo chamado ELM 327 que vai conectado na saída OBD II do veículo analisado.

Por fim, cita-se Cavalcante (2018), que produziu sua própria placa eletrônica que conectada ao veículo por um cabo leitor de OBD II adquiria parâmetros e registros de falhas e os enviava via Bluetooth para um celular.

4. OBJETIVO

Nos dias atuais, busca-se a melhor eficiência dos recursos energéticos, sendo assim, existe uma grande preocupação social para diminuir o consumo de gasolina, etanol e diesel por parte dos veículos. Entretanto, a grande maioria dos motoristas desconhece como conduzir com a melhor economia de combustível, uma vez que seus veículos não apresentam de maneira simples as variáveis necessárias. Sendo assim, o presente trabalho objetiva captar parâmetros de funcionamento de um veículo que não seja de luxo para responder a dúvida presente dos condutores: “qual a maneira de condução com maior eficiência do combustível?”.

5. FUNDAMENTAÇÃO TEÓRICA

No decorrer deste capítulo apresentam-se fundamentos importantes para a correta compreensão do trabalho.

5.1. ARDUINO MEGA 2560

De acordo com a fabricante Atmel, o Arduino Mega 2560 é uma placa de prototipagem eletrônica que apresenta um processador ATmega2560 que opera numa tensão elétrica de 5V (Arduino, 2019). Esse micro controlador opera de maneira semelhante ao seu antecessor, o Arduino Uno, mas difere pelo fato de apresentar um maior número de entradas e saídas para comunicação e na velocidade com que transmite as informações.

O Arduino Mega 2560 tem sido preferência entre os programadores devido a seu baixo custo e o seu fácil acesso no mercado. Entretanto, o principal ponto positivo desta placa é o seu ambiente de programação em linguagem Java, disponibilizado de maneira gratuita no site da fabricante, o qual faz comunicação com a placa via entrada USB.

5.2. REDE CAN

A rede CAN trata-se de modelo de funcionamento de uma central eletrônica veicular que foi inventado por Robert Bosch em 1980. De acordo com Bosch (2005), a grande vantagem da Rede CAN foi o seu método de barramento de dados que permitiu numa redução significativa do volume de cabos nos veículos e da probabilidade de falhas na rede de aparelhos. A rede CAN conseguiu tal avanço utilizando identificadores em cada mensagem, que permitem avaliar o conteúdo e a sua prioridade. Desta forma, a rede CAN baseia-se em dois cabos que enviam mensagens de maneira espelhada (polaridades invertidas), o que minimiza o efeito de campos eletromagnéticos e a interferência de ruídos na rede.

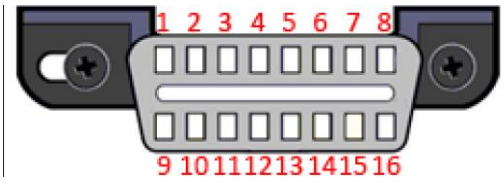
Apesar de a rede CAN manter seu conceito de funcionamento, ela possui diferentes padrões que seguem determinadas normas e variam de acordo com o tipo do equipamento. Por exemplo: tratores e veículos agrícolas seguem a ISO 11783 ou a DIN 9684, enquanto os veículos de passeio são orientados pela ISO 11898 ou SAE J1939.

5.3. OBD II

Conforme Bosch (2005), a sigla OBD (*On Board Diagnostics*) é sistema de diagnóstico de bordo que monitora o funcionamento do motor. A sua primeira fase (OBD I) foi desenvolvida em 1988 na Califórnia/EUA, com o intuito de monitorar os gases do escapamento dos automóveis. Já em 1994 foi produzido o OBD II, que é a segunda geração deste sistema de diagnóstico embarcado. De acordo com Machado e Oliveira (2007), essa nova versão trouxe diversos avanços, uma vez que permitiu a análise de um número maior de parâmetros e padronizou os protocolos e métodos de conexão entre todas as montadoras.

A padronização de leitura é realizada por meio de um conector com 16 pinos, numerados conforme a Figura 3, de acordo com as especificações da norma SAE J1962. Desde 2010 no Brasil, todos os veículos são obrigados a possuírem este conector. Entretanto a sua posição no veículo pode variar de acordo com o modelo do automóvel e os pinos presentes em cada entrada podem ser diferentes em função do protocolo utilizado. Por exemplo, de acordo com a Elm Electronics Inc. (2019), automóveis que seguem a norma da rede CAN 2.0 precisam ter o pino 6 e o 14, enquanto os da ISO 9141-2 obrigatoriamente devem apresentar os pinos 7 e 15.

Figura 3 – Numeração dos 16 pinos no conector OBD II.



Fonte: www.obddiag.net/adapter.html

5.4. ELM 327

Conforme exposto pela fabricante Elm Electronics Inc. (2019), o equipamento ELM 327 é um dispositivo desenvolvido para interpretar dados da rede CAN via a entrada OBD II e os enviar por conexão Bluetooth. O ELM 327 teve sua primeira versão desenvolvida em 2005 e desde então vem sendo otimizado. Essa constante atualização permitiu a última versão do equipamento estabelecer comunicação com a maioria dos protocolos de rede CAN presentes no mercado, sendo eles: SAE J1850, ISO 9141-2, ISO 14230-4, ISO 15765-4, SAE J2411 e SAE J1939. A Figura 4 apresenta o dispositivo ELM 327 utilizado no presente trabalho.

Figura 4 – Dispositivo ELM 327 utilizado no presente trabalho.

5.5. NORMA SAE J1939

A norma SAE J1939 (2006) tem sua aplicação para rede CAN automotiva, tendo sido inicialmente voltada para veículos pesados, como ônibus e caminhões. A norma é também bastante aplicada pelas fabricantes automotivas para veículos de passeio, o que a fez ser escolhida para ser utilizada no presente trabalho. Esta norma apresenta o modo de identificação dos serviços e parâmetros de diagnósticos, assim como o modo de interpretação dos dados transmitidos pela rede CAN.

No presente trabalho optou-se pelo serviço $01, que faz a requisição atual de parâmetros identificados por PID (*Parameter IDentifier*, em inglês, ou Identificador do Parâmetro). Os PIDs selecionados para serem avaliados foram o 04,05, 0C, 0D, 2F e 5E, que respectivamente referem-se a carga calculada do motor, temperatura do líquido de arrefecimento, rotação do motor, velocidade do veículo, nível de combustível e o consumo de combustível. Dentre os parâmetros escolhidos, somente a rotação e o consumo são fornecidos pela rede CAN via dois bytes (A e B), enquanto os demais são por um byte (A) Todos os bytes lidos apresentavam a mesma variação que é de 0 a 255.

5.6. CARGA CALCULADA DO MOTOR

Conforme a norma SAE J1939 (2006), a carga calculada do motor indica a razão da vazão de ar na admissão pelo volume máximo de entrada de ar para determinada rotação

quando o pedal de aceleração está totalmente acionado. Este valor é corrigido pela altitude e temperatura ambiente.

Em outras palavras, este parâmetro representa o percentual de abertura da borboleta do motor. O valor fornecido pela rede CAN para este PID 04, precisa ser corrigido conforme a Equação 1 e pode oscilar de 0% até 100%.

$$Cm = A(\frac{100}{255}) \quad (1)$$

onde Cm é o valor da carga calculada do motor (%) e A é o byte fornecido pela rede CAN (adimensional).

5.7. TEMPERATURA DO LÍQUIDO DE ARREFECIMENTO

Conforme a norma SAE J1939 (2006), a temperatura do líquido de arrefecimento é medida por um sensor térmico posicionado diretamente no fluido de arrefecimento ou na cabeça do cilindro do motor. Este parâmetro serve para monitorar se a temperatura de operação do motor está adequada. O valor fornecido pela rede CAN para este PID 05, precisa ser corrigido conforme a Equação 2 e pode oscilar de -40°C até 210°C.

$$T = A - 40 \quad (2)$$

onde T é o valor da temperatura do líquido de arrefecimento (°C) e A é o byte fornecido pela rede CAN (adimensional).

5.8. ROTAÇÃO DO MOTOR

Conforme a norma SAE J1939 (2006), a rotação do motor indica o número de revoluções que a árvore de manivelas realiza em um minuto. Este parâmetro é um dos principais parâmetros para caracterizar a maneira como veículo é conduzido. O valor fornecido pela rede CAN para este PID 0C, precisa ser corrigido conforme a Equação 3 e pode oscilar de 0 rpm até 16383,75 rpm.

$$Rot = \frac{A256 + B}{4} \quad (3)$$

onde Rot é o valor da rotação do motor (rpm), A é o byte primário fornecido pela rede CAN (adimensional) e B é o byte secundário fornecido pela rede CAN (adimensional).

5.9. VELOCIDADE DO VEÍCULO

Conforme a norma SAE J1939 (2006), a velocidade do veículo é obtida através de um cálculo utilizando sensores do sistema de transmissão. O valor fornecido pela rede CAN para este PID 0D, diferentemente dos outros, não precisa ser corrigido segundo a norma SAE J1939, consequentemente ele pode oscilar de 0 km/h até 255 km/h.

5.10. NÍVEL DE COMBUSTÍVEL

Conforme a norma SAE J1939 (2006), o nível de combustível indica o percentual de combustível em relação à capacidade máxima. Este parâmetro pode ser obtido diretamente de um sensor de nível no tanque do veículo ou inferido de maneira indireta via a leitura de outros sensores. Entretanto, não são todos os veículos que apresentam comunicação entre o nível de combustível e a rede veicular. O valor fornecido pela rede CAN para este PID 2F precisa ser corrigido conforme a Equação 4 e pode oscilar de 0% até 100%.

$$NComb = A(\frac{100}{255}) \quad (4)$$

onde $NComb$ é o valor do nível do combustível (%) e A é o byte fornecido pela rede CAN (adimensional).

5.11. CONSUMO DE COMBUSTÍVEL

Conforme a norma SAE J1939 (2006), o consumo de combustível indica quanto será consumido em um intervalo de uma hora se o veículo continuar sendo utilizado sem sofrer variações. Entretanto, assim como o nível de combustível, este parâmetro não é presente em todos os veículos que seguem a tal normal. O valor fornecido pela rede CAN para este PID 5E, precisa ser corrigido conforme a Equação 5 e pode oscilar de 0 l/h até 3276,75 l/h.

$$Cons = \frac{(A256 + B)5}{100} \quad (5)$$

onde $Cons$ é o valor do consumo de combustível (l/h), A é o byte primário fornecido pela rede CAN (adimensional) e B é o byte secundário fornecido pela rede CAN (adimensional).

6. METODOLOGIA

A metodologia consistiu em duas etapas: desenvolvimento do equipamento e na execução dos testes de rodagem no veículo. No decorrer deste capítulo descreve-se tais passos, assim como as dificuldades encontradas.

6.1. DESCRIÇÃO DO EQUIPAMENTO

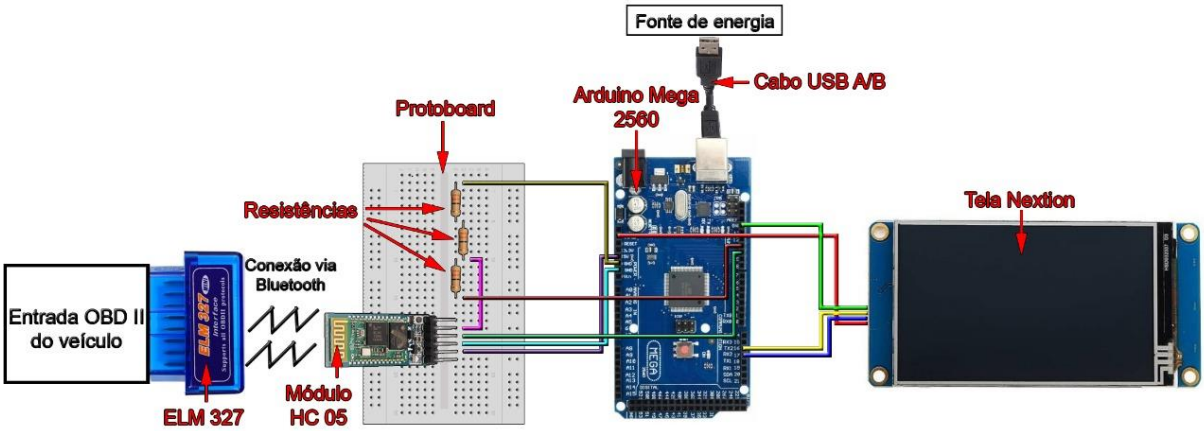
Para o desenvolvimento do equipamento, adquiriu-se um Arduino Mega 2560, um ELM 327, um módulo HC 05, uma tela LCD Nextion (3,5 polegadas 480x320, modelo Nx4832t035), uma placa protoboard, três resistores de 33 ohms (tolerância +/- 5%), dez jumpers do tipo macho-macho e um cabo USB A/B. A função de cada componente no equipamento montado pode ser verificado na Tabela 1.

Tabela 1 – Função de cada componente no funcionamento do equipamento desenvolvido.

Componente	Função
Arduino Mega 2560	• Controlar os outros componentes; • Receber a implementação do código; • Executar a correção dos dados da rede CAN.
ELM 327	• Realizar a conexão com o veículo via o conector OBD II; • Interpretação dos dados da rede CAN.
Módulo HC 05	• Fazer a conexão via bluetooth do ELM 327 com o Arduino Mega 2560.
Tela LCD Nextion	• Impressão dos valores corridos dos parâmetros lidos do veículo.
Placa protoboard	• Fazer a conexão entre o Arduino, resistores e módulo HC 05.
Resistores 33 ohms	• Criar um divisor de tensão para o Arduino e o módulo HC 05 se comunicar sem queimar os componentes, pois funcionam em voltagens distintas.
Jumpers	• Interligar os componentes do circuito.
Cabo USB A/B	• Conectar o Arduino com o computador para implementar o código; • Conectar o equipamento com a fonte de energia; • Conectar o equipamento com o computador para impressão e registro de todos os dados lidos durante os testes.

Na Figura 5 pode-se verificar a configuração dos equipamentos, assim como as conexões entre eles via os jumpers que estão sendo representados pelas linhas coloridas.

Figura 5 – Disposição dos componentes com o equipamento montado.



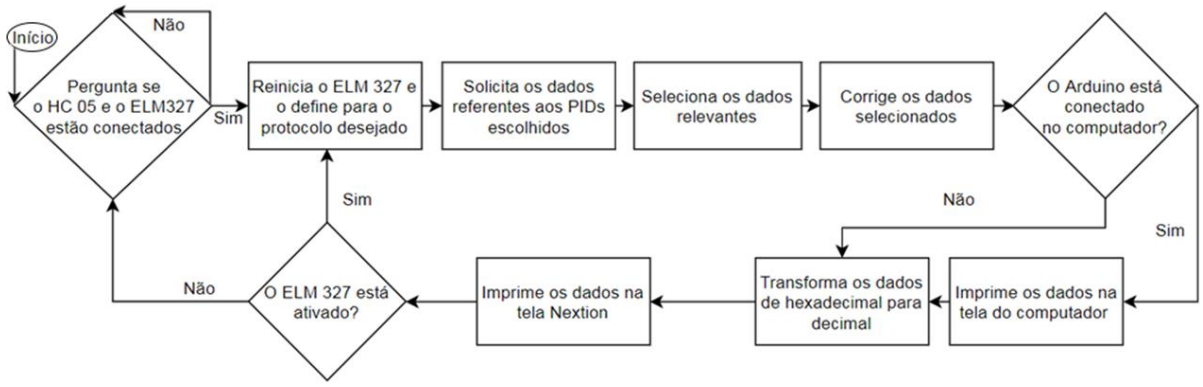
Para o desenvolvimento do código implementado no Arduino, inicialmente determinou-se que seriam lidos os parâmetros de uma rede CAN que seguia a norma SAE J1939. Justifica-se esta escolha pelo fato dele ser utilizada por grande parte das maiores montadoras de veículos do mundo, como é o caso da Volkswagen, Hyundai, Citröen, entre outras. Consequentemente, isso facilitou o acesso a veículos com tal protocolo para a execução dos testes.

Já definida a norma a ser seguida, efetuou-se a conexão entre o módulo HC 05 e o ELM 327 para quando ativados parearem-se automaticamente via Bluetooth e transmitissem informações numa velocidade de 9600 baud (bits/segundos), a qual é compatível com a rede CAN e o Arduino Mega 2560.

Com os equipamentos configurados, desenvolveu-se o código em Java que executou a seguinte rotina: configurar o ELM 327 para operar no protocolo desejado; solicitar as respostas dos parâmetros escolhidos a serem lidos via os PID 04,05, 0C, 0D, 2F e 5E (que respectivamente referem-se a carga calculada do motor, temperatura do líquido de arrefecimento, rotação do motor, velocidade do veículo, nível de combustível e o consumo de combustível); identificar os valores respondidos; corrigir os valores e imprimir no computador (se conectado); fazer a conversão dos valores de hexadecimal para decimal em função da modo de funcionamento de impressão dos valores na tela do equipamento. Este resumo é executado de modo cíclico da etapa de solicitação dos parâmetros até a impressão na tela do equipamento, sendo o período demandado pelo código de aproximadamente 2 segundos. Destaca-se que o programa criado não causa risco nenhum ao veículo analisado, uma vez que somente executa o procedimento de leitura, ou seja, não modifica a operação da central eletrônica.

Para exemplificar o funcionamento do código, apresenta-se na Figura 6 o algoritmo programado em diagrama de blocos.

Figura 6 – Diagrama de blocos do algoritmo programado.



6.2. PROBLEMAS ENCONTRADOS

Após a construção e programação do equipamento verificaram-se determinados problemas. O primeiro deles foi que a programação ficou extensa, o que resultou em um intervalo de tempo de 2 segundos entre a leitura do parâmetro até a sua impressão digital. O principal responsável foram os atrasos estipulados no código para obtenção dos dados, uma vez que se precisa alinhar a velocidade de transmissão das informações. Visto que a informação da rotação do eixo de manivelas é fornecida a cada 500 ms, enquanto as outras são a cada 100 ms, o que resulta em um atraso total de 1 segundo. O restante do atraso é devido ao tempo de processamento dos dados pelo Arduino e pela tela, juntamente com a demora em transmitir via Bluetooth. A maneira mais simples para reduzir estes atrasos seria utilizando equipamentos com processadores mais rápidos.

O segundo problema foi a dimensão da tela do equipamento que permitiu expor apenas 5 dos 6 parâmetros lidos. A adição de mais uma medida resultava na redução do tamanho da fonte exibida, o que poderia causar uma distração devido a maior dificuldade para o motorista ler os valores. Por motivos de segurança, escolheu-se expor um menor número de parâmetros. Entretanto, para não omitir um parâmetro lido, acrescentou-se no código a possibilidade para expor na tela o nível de combustível ou o consumo de acordo com o desejo do usuário, juntamente das outras medidas que são fixas.

O terceiro problema foi que nem todos os veículos apresentam a leitura do nível e do consumo de combustível. Destaca-se que o problema não era na leitura, uma vez que a resposta fornecida pela central eletrônica era "*no data*" (sem dados). A dificuldade é que consumo somente é apresentado em veículos com maior tecnologia e consequentemente de uma classe veicular acima dos automóveis populares, os quais não eram o alvo de análise deste trabalho. Já o nível combustível era lido em automóveis populares, mas de uma maneira indireta, por inferência, o que fazia com que seu valor não sofresse alterações instantâneas, possuindo baixa exatidão.

6.3. METODOLOGIA DOS TESTES

Para executar os testes, selecionou-se um veículo Volkswagen Gol Mb5, ano 2019, modelo 2019, com motor de 1,6 litros de volume, cuja rede CAN seguia o protocolo da norma SAE J1939. A dificuldade encontrada foi que este automóvel não indicava o consumo instantâneo e a leitura do seu nível de combustível não apresentava exatidão. Verificou-se esta característica ao completar o tanque com 3 litros de etanol (5,45% do volume máximo do tanque de 55 litros) e não ocorrer à variação de nem 1 % na medida lida pelo equipamento.

Sendo assim, para ter a medida do consumo total em litros, optou-se por um trajeto de testes iniciando e terminando num posto de gasolina, onde o tanque do automóvel era completado com etanol até o primeiro estalo pela mesma bomba de combustível antes de começar cada teste. Ressalta-se que o valor lido nas bombas para completar o tanque após todos os testes possuem um imprecisão de 3 mililitros, visto que elas eram certificadas pelo Inmetro conforme a Portaria nº 294, de 29/06/2018.

O trajeto percorrido em cada teste foi em um trecho urbano de aproximadamente 7,7 km com limite de velocidade de 60 km/h. O tempo médio para percorrê-lo é de 14 minutos, segundo o Google Maps (Google, 2019). O percurso apresentava trechos em aclive, declive, retas e ainda 25 semáforos. A fim de minimizar a influência do trafego, realizaram-se os testes durante a madrugada devido ao menor tráfego de automóveis. Na Figura 7 expõe-se a posição de início e fim dos testes, assim como o trajeto percorrido pelas Av. Carlos Gomes, Av. Senador Tarso Dutra e Av. Ipiranga no município de Porto Alegre/RS.

Figura 7 – Trajeto percorrido nos testes.

Fonte: Google Maps

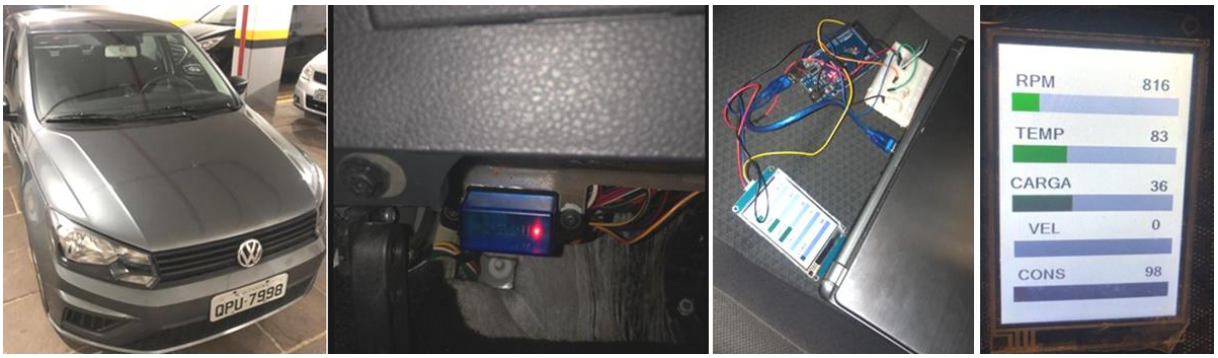
Os testes consistiram em um mesmo motorista percorrer o trajeto variando o modo de condução sem infringir qualquer lei de trânsito. Os modos de condução sempre visavam permanecer na maior marcha possível, mas variaram pela rotação na qual elas eram trocadas, sendo os limites de corte de 2000 rpm, 3000 rpm e 4000 rpm. Além disso, para cada limite de rotação foram feitos dois testes: um deles buscando o acionamento total do acelerador e o outro com pouco acionamento (menos de metade do curso máximo). Portanto, executaram-se um total de 6 testes, que tem suas diferenças expostas na Tabela 2.

Tabela 2 – Parâmetros de cada teste executado.

Teste #	Rotação máxima atingida	Acionamento do acelerador
1	2000 rpm	Pouco (menos da metade do curso máximo)
2	3000 rpm	Pouco (menos da metade do curso máximo)
3	4000 rpm	Pouco (menos da metade do curso máximo)
4	2000 rpm	Completo
5	3000 rpm	Completo
6	4000 rpm	Completo

Na Figura 8 observa-se, da esquerda para a direita, o veículo utilizado nos testes, o ELM 327 conectado na entrada OBD II, o equipamento conectado no computador para registro dos valores e a tela com os valores medidos.

Figura 8 – Registros durante a execução dos testes.



7. RESULTADOS E ANÁLISE

Realizados os testes, obtiveram-se os valores gerais expressos na Tabela 3.

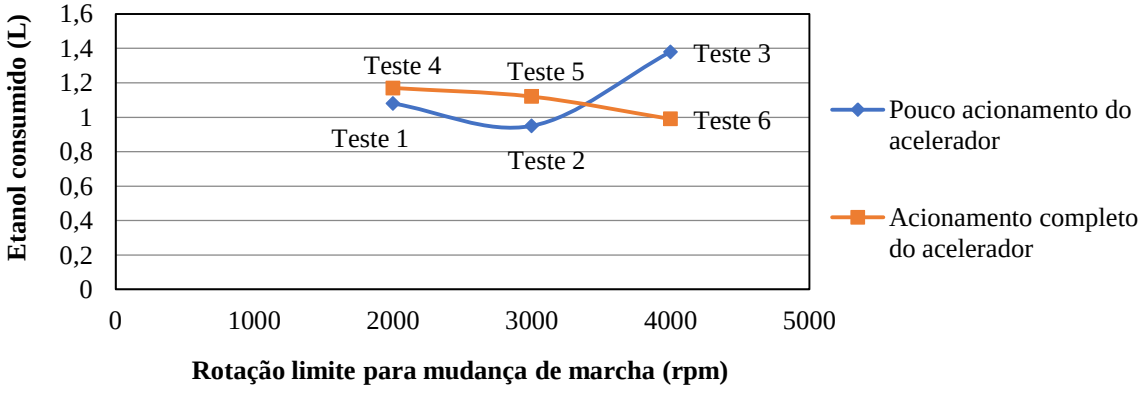
Tabela 3 – Valores gerais médios e totais obtidos nos testes.

Teste	Velocidade média (km/h)	Temperatura média do líquido de arrefecimento (°C)	Carga calculada média do motor (%)	Média das rotações (rpm)	Consumo total de etanol (L)	Duração (min:s)	Distância percorrida em função consumo (km/l)
1	39,76	81,10	40,62	1497,88	1,08	11:16	7,12
2	43,51	80,82	33,75	2132,65	0,95	10:18	8,10
3	37,67	78,58	33,30	2156,65	1,38	11:54	5,57
4	41,46	81,27	43,25	1620,97	1,17	10:48	6,58
5	41,63	80,79	34,71	2165,72	1,12	10:48	6,87
6	40,08	79,82	33,87	2177,63	0,99	11:11	7,77

Fazendo uma macroanálise dos testes, pode-se atribuir a influência da presença dos semáforos, caso observados os valores gerais, visto que quando fechado é preciso parar do o veículo, fazendo diminuir a média da velocidade, da carga calculada do motor e das rotações e aumentar o tempo de duração dos testes.

Analisando-se os valores do consumo total de etanol em função da rotação de corte para mudança de marcha em cada teste e dividindo-se pelo nível de acionamento do acelerador, resultando nas informações contidas na Figura 9.

Figura 9 – Etanol consumido *versus* rotação limite para mudança de marcha.



Verifica-se que com pouco acionamento do acelerador a maneira de condução que proporciona o ponto de menor consumo é aquela onde as mudanças de marcha são realizadas com aproximadamente 2700 rpm, pelo fato de ser o ponto mais inferior da curva azul traçada na Figura 9.

Já para um modo de condução com acionamento completo do acelerador, não se identifica o ponto nominal à primeira vista, uma vez que a quantidade de etanol consumido está diminuindo. Entretanto, observado o comportamento da curva de consumo para pouca aceleração como uma parábola positiva, pode-se aferir que para uma condução com elevada aceleração, o ponto ótimo para realizar a troca de marchas é próximo a 4000 rpm.

Apresentados os resultados gerais, volta-se a atenção para cada parâmetro lido.

7.1. ANÁLISE DA VELOCIDADE

Para facilitar a visualização dividem-se as velocidades obtidas ao longo dos testes na Figura 10 e na Figura 11.

Figura 10 – Velocidades nos testes com pouca aceleração *versus* tempo.

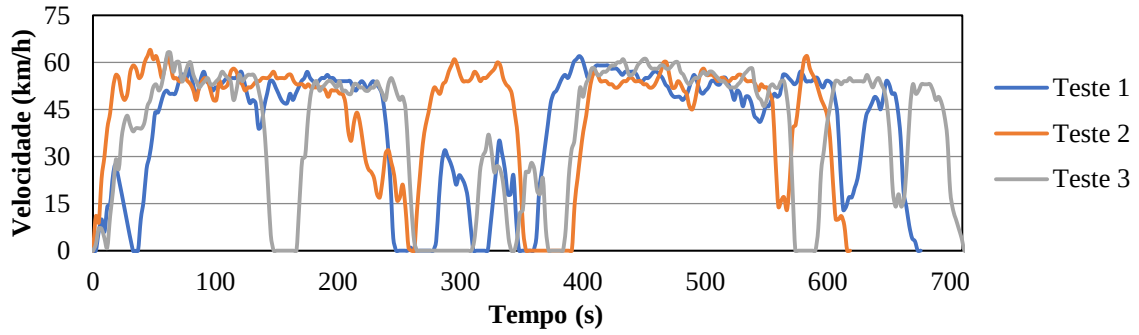
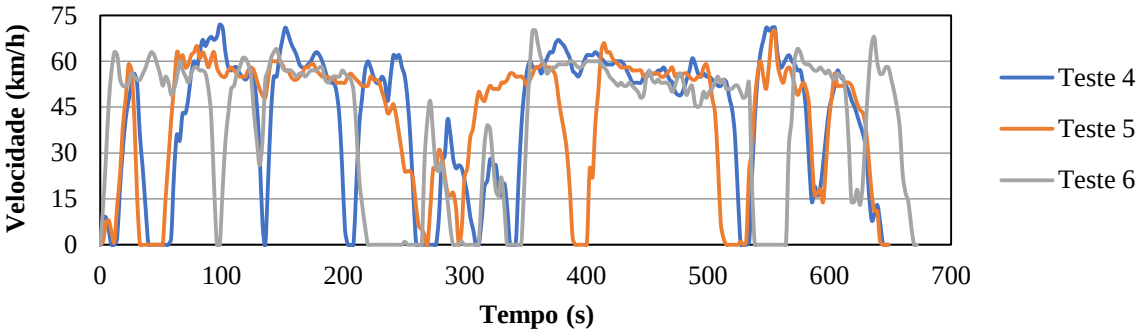


Figura 11 – Velocidades nos testes com aceleração total *versus* tempo.



O comportamento da velocidade permite justificar a variação na duração dos testes como sendo em função dos semáforos fechados, porque os vales até 0 km/h significam sinais vermelhos. Sendo assim, nota-se que a variação da duração dos testes é consequência diretas do fechamento dos semáforos.

Ressalta-se que foram raras as ocasiões em que não foi respeitado o limite de velocidade da metodologia dos 60 km/h. Ao tomar esse cuidado com a velocidade, observou-se que o valor apresentado no painel do veículo conservava cerca de 5 km/h a mais em relação ao do equipamento. Essa é uma incerteza projetada pelas fabricantes para evitar problemas judiciais devido a multas.

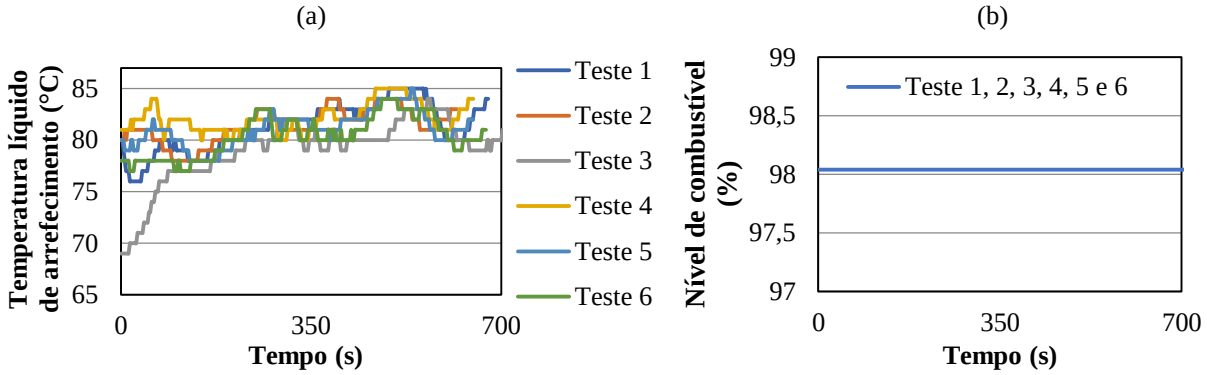
7.2. ANÁLISE DO NÍVEL DE COMBUSTÍVEL E DA TEMPERATURA DO LÍQUIDO DE ARREFECIMENTO

A temperatura do líquido de arrefecimento e o nível de combustível não trouxeram grandes contribuições para este trabalho.

Os valores lidos para temperatura se mantiveram dentro de uma faixa de operação de baixa variação com exceção do teste 3, conforme apresentado na Figura 12 (a). O teste 3 apresenta essa diferença, porque foi o primeiro a ser executado e o automóvel estava parado antes de executá-lo. Entretanto, como indica o teste 3, bastaram 3 minutos de teste para atingir-se o valor praticamente estável de 80°C.

Já o nível de combustível não contribuiu com análise, porque apresentou em todos os testes o valor constante de 98,04%, conforme pode ser visto na Figura 12 (b).

Figura 12 – (a) Temperatura do líquido de arrefecimento *versus* tempo. (b) Nível de combustível *versus* tempo.



7.3. ANÁLISE DA CARGA CALCULADA DO MOTOR

Assim como o realizado para as velocidades, dividiram-se as cargas calculadas do motor obtidas ao longo dos testes na Figura 13 e na Figura 14 para facilitar a visualização. Desta forma, pode-se verificar que para aceleração total conseguiu-se obter um número maior de picos nos 100 %, conforme o desejado. Já com pouca aceleração, só houve picos próximos aos 600 segundos em função da necessidade imposta pelo perfil da pista que era a área de aclive e se evitava a redução de marchas para seguir a metodologia adotada.

Figura 13 – Velocidades nos testes com pouca aceleração *versus* tempo.

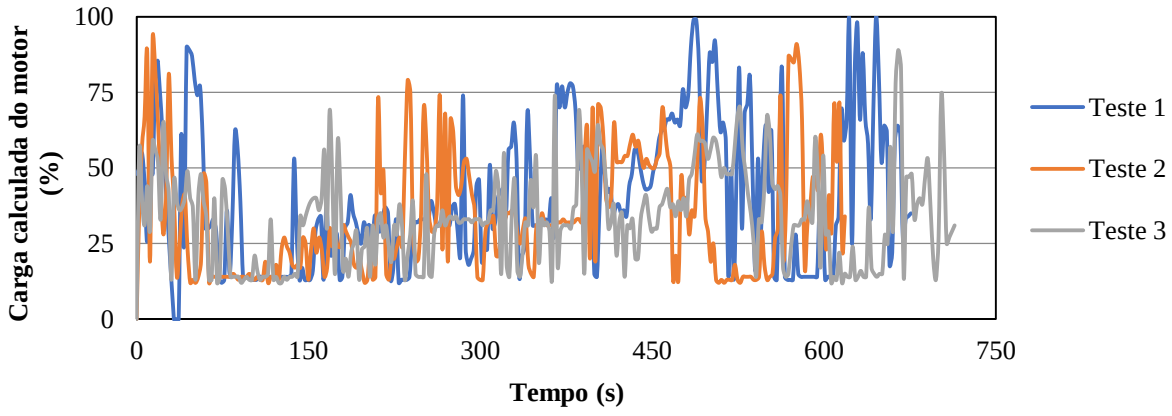
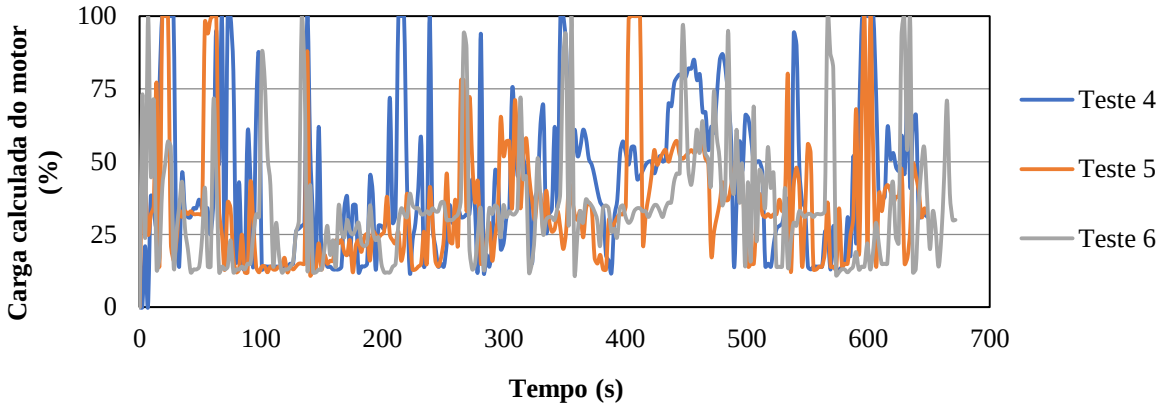


Figura 14 – Velocidades nos testes com aceleração total *versus* tempo.

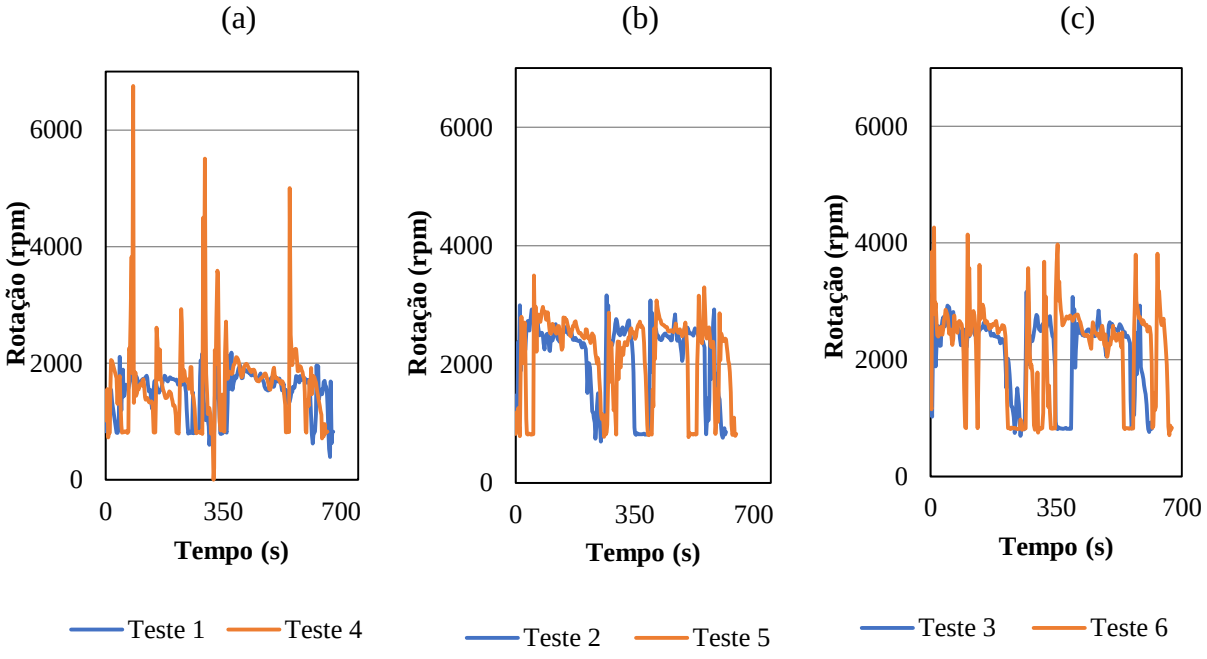


Além disso, comparando a Figura 13 com a Figura 10 e a Figura 14 com Figura 11 identificou-se que o automóvel demanda cerca de 30% de sua carga máxima para permanecer em funcionamento quando se desloca sem grandes variações de velocidade ou está parado (0 km/h).

7.4. ANÁLISE DA ROTAÇÃO

Sobre a variação das rotações durante os testes, verifica-se que foi respeitada, na maioria do tempo de duração, a metodologia para mudanças de marchas. A única exceção foi durante o teste 4, no qual ocorreram situações de corte de giro devido a falha humana decorrente da dificuldade de mudar do câmbio sem superar as 2000 rpm. Além disso, constata-se que os comportamentos dos gráficos em cada teste possuem certa repetibilidade, o que indica que os procedimentos de arrancada são semelhantes.

Figura 15 – (a) Rotação com mudança de marcha antes dos 2000 rpm *versus* tempo.
(b) Rotação com mudança de marcha antes dos 3000 rpm *versus* tempo.
(c) Rotação com mudança de marcha antes dos 4000 rpm *versus* tempo.



8. CONCLUSÕES

Este trabalho apresenta a análise da influência do modo de condução em um veículo de passeio via a leitura da Rede CAN 2.0 com o uso de Arduino. A funcionalidade da rede CAN fornecer parâmetros de operação após o desenvolvimento de um computador de bordo portátil foi evidenciada. Assim, pode-se ver com mais facilmente a influência de condução de um automóvel de acordo com o seu modo de operação.

A partir deste estudo, concluem-se diversos aspectos. Percebeu-se a necessidade de um aprimoramento na leitura do nível de combustível em carros populares por parte das montadoras, devido ao fato de muitos deles não informarem tal dado e aqueles que apresentavam, o fazem com baixa exatidão. Esse avanço é essencial para que os condutores tenham a possibilidade de buscar a maneira mais adequada de condução.

Outro aspecto apurado é que a temperatura do líquido de arrefecimento não é um parâmetro que apresenta significativas alterações devido à forma de condução empregada neste trabalho, mas pode-se aferir o pouco intervalo de tempo demandado para este parâmetro atingir a estabilidade.

Já a carga calculada do motor possibilitou a aferição de que o veículo demanda aproximadamente 30% do seu valor máximo para se manter em inércia em situações de funcionamento. Além disso, notou-se que essa carga mínima tem extrema representatividade nos resultados, pois esta é a situação em que o motor do veículo é mais requisitado em uma utilização urbana com baixas acelerações e muito tempo parado em semáforos, ou seja, um projeto para reduzi-la aumentaria a eficiência do consumo de combustível.

Por fim, utilizando os parâmetros lidos via a rede CAN (velocidade, rotação e carga do motor calculada) para controlar o veículo durante os testes, conseguiu-se apurar a melhor maneira de condução e consequentemente, atingir o objetivo do presente trabalho. Conclui-se que o ponto de menor consumo pode variar conforme a intensidade na qual o acelerador é pressionado, sem sofrer grandes variações. Visto que com um maior acionamento do pedal do acelerador, o ponto de melhor eficiência do combustível desloca-se para a troca de marchas com maiores rotações. Entretanto, pode-se indicar uma melhor maneira de condução como sendo pouca aceleração junto da troca de marchas na faixa das 2700 rpm. Esta maneira de condução se sobressai mesmo sem ter grandes variações no nível de combustível consumido em relação à condução com maior aceleração, pois ao operar em menores rotações, os componentes do motor do veículo são submetidos a um menor número de ciclos que resulta em menor desgaste.

Tendo em mente a busca mundial pela redução da poluição e a melhor utilização dos recursos energéticos, percebe-se a importância do assunto e da necessidade de um aprofundamento, visto os resultados obtidos no consumo total para um mesmo trajeto, sendo o maior valor (1,38 litros) 45% superior em relação ao menor (0,9 litros). Desta forma, sugere-se que sejam realizados outros trabalhos na linha para encontrar a melhor maneira de condução dos automóveis, como por exemplo:

- Avaliar a influência da maneira de condução em relação a outros parâmetros monitorados pela rede CAN;
- Analisar a curva de consumo utilizando outros parâmetros de controle do modo de condução.

REFERÊNCIAS BIBLIOGRÁFICAS

ALMEIDA, Rodrigo. **Sistema de análise de consumo de combustível de veículos automotores**. 2017. Trabalho de conclusão (Ciência da computação) - Instituto federal Minas Gerais, Formiga, 2017.

ANP. **Anuário estatístico brasileiro do petróleo, gás natural e biocombustíveis**, 2018. Disponível em: <http://www.anp.gov.br/images/central-de-conteudo/publicacoes/anuario-estatistico/2018/anuario-2018-versao-impressao.pdf>. Acesso em: 22 out. 2019.

ARDUINO. **Compare Board Specs**. [S. .l.], 2019. Disponível em: <https://www.arduino.cc/en/Products/Compare> Acesso em: 10 set. 2019.

BMW. **Computador de bordo: Todas as informações de imediata**. [S. l.], 2016. Disponível em: <https://www.bmw-motorrad.com.br/pt/experience/engineering/detail/navigation-communication/onboardcomputer.html>. Acesso em: 27 set. 2019.

BOSCH. **Leitores OBD**. [S. l.], 2019. Disponível em: <https://www1.bosch.com.br/testequipment/products?gclid=Cj0KCQiA-4nuBRCnARIsAHwyuPoE1zxQaOJsF9eNvt3xQVZBZorHmm7wubd7m2KXf4tom1E7ansR6g4aAtksEALw_wcB#obd-readers>. Acesso em: 27 set. 2019.

BOSCH, Robert. **Manual de Tecnologia Automotiva**. Tradução de Helga Madjderey, Gunter W. Prokesch, Euryale de Jesus Zerbini, Sueli Pfeferman. 25° ed. São Paulo: Edgar Blücher, 2005.

CAVALCANTE, Larissa. **Sistema de monitoramente automotivo via rede CAN**. 2018. Trabalho de conclusão (Engenharia mecatrônica) - Instituto federal de educação, ciência e tecnologia de Santa Catarina, Florianópolis, 2018.

CHEVROLET. **Manual do proprietário Chevrolet Tracker**. [S. l.], 2013. Disponível em: <https://www.chevrolet.com.br/content/dam/chevrolet/mercosur/brazil/portuguese/index/services/owner-manuals/05-pdf/tracker/14tracker-brazil-pt-br-083013-v6-baixa.pdf>. Acesso em: 27 set. 2019.

ELM ELECTRONICS INC. **OBD Help**. [S. l.], 2019. Disponível em: <https://www.elmelectronics.com/help/obd>. Acesso em: 10 set. 2019.

ELM ELECTRONICS INC. **OBD: Intrerpreter ICs**. [S. l.], 2019. Disponível em: <https://www.elmelectronics.com/products/ics/obd/> Acesso em: 10 set. 2019.

FARINELLI, Felipe. **Desenvolvimento de um sistema embarcado de bordo para monitoramento de parâmetros via diagnóstico de rede CAN**. 2016. Trabalho de conclusão (Engenharia eletrônica) - Universidade tecnológica federal do Paraná, Ponta Grossa, 2016.

GOOGLE, Google Maps. [S. l.], 2019. Disponível em: <https://www.google.com.br/maps/preview>. Acesso em: 05 nov. 2019.

HONDA. **Manual do proprietário Honda Civic**. [S. l.], 2018. Disponível em: <https://www.honda.com.br/pos-venda/automoveis/sites/customer_service/files/2018-

02/Civic%202018%20-%20Manual%20do%20propriet%C3%A1rio_20180123.pdf>. Acesso em: 27 set. 2019.

IVECO. **Hi-Technology & telematics: Computador de bordo**. [S. l.], 2019. Disponível em: <https://www.iveco.com/brasil/produtos/pages/hiway_tecno_telem_01.aspx>. Acesso em: 27 set. 2019.

MACHADO, A. S. L.; OLIVEIRA, B. R. R. **O sistema obd (on-board diagnosis)**. ISEP, Porto - Portugal, 2007.

MINISTÉRIO DO MEIO AMBIENTE. **Relatório final do 1° inventário nacional de emissões atmosféricas por veículos automotores rodoviárias**. [S. l.], 2011. Disponível em: <https://www.mma.gov.br/estruturas/163/_publicacao/163_publicacao27072011055200.pdf>

MULTILASER. **OBD II carrorama app android conexão ecu via bluetooth - AU205**. [S. l.], 2019. Disponível em: <https://www.multilaser.com.br/scanner-automotivo-bluetooth-obdii-carrorama-by-multilaser-au205/p>. Acesso em: 27 set. 2019.

RAVEN. **Conheça o Scanner 3**. [S. l.], 2019. Disponível em: <https://www.ravenscanner3.com.br/sobre>. Acesso em: 27 set. 2019.

SAE. **J1979: E/E Diagnostic Test Modes. 2006.** Detroit: Sae, 2006.

SINDIPEÇAS. **Relatório da frota circulante**. . [S. l.], 2019. Disponível em: <https://www.sindipecas.org.br/sindinews/Economia/2019/RelatorioFrotaCirculante_Maio_2019.pdf> Acesso em: 22 out.. 2019.

ZANLORENSSI, Gabriel; ALMEIDA, Rodolfo. **A trajetória do preço do combustível no Brasil nos últimos 17 anos**. [S. l.], 16 out. 2017. Disponível em: <https://www.nexojornal.com.br/grafico/2017/10/16/A-trajet%C3%B3ria-do-pre%C3%A7o-do-combust%C3%ADvel-no-Brasil-nos-%C3%BAltimos-17-anos> Acesso em: 22 out. 2019.

APÊNDICE – Código em Arduino utilizado neste trabalho para leitura da rede CAN

Base computador de bordo

```
#include <SoftwareSerial.h>
#include  "Nextion.h"
int valor=0;String mostra_consumo="";
String mostra_carga="";
String mostra_rpm="";
String mostra_velocidade="";
String mostra_temperatura="";
String mostra_nivel="";
String mostra_total="";
int y=0,t=0,c=0,v=0,cons=0;
SoftwareSerial bluetooth_serial(10, 11);
#define CMD_RPM        0
#define CMD_TMP        1
#define CMD_VEL        2
#define CMD_ACEL       3
#define CMD_COMBUSTIVEL 4
#define CMD_BATeria   5
#define CMD_CARGA_MOTOR 6
#define    pot    A0
#define    led    13
int     pot_value = 0,
        bar_value = 0;
char    txt1[10],
        txt2[10];
uint32_t ds_var;
int     rpm_value = 0,
        rpm_bar_value = 0;
    NexProgressBar j0 = NexProgressBar(0,
3, "j0");
    NexText rpm_val   = NexText(0, 1, "t0");
int  temperatura_value = 0,
temperatura_bar_value =0;
    NexProgressBar j1 = NexProgressBar(0,
4, "j1");
NexText temperatura_val   = NexText(0, 8,
"t2");
int     carga_motor_value = 0,
    carga_motor_bar_value = 0;
    NexProgressBar j2 = NexProgressBar(0,
5, "j2");
    NexText carga_motor_val   = NexText(0,
10, "t4");
int     velocidade_value = 0,
    velocidade_bar_value = 0;
    NexProgressBar j3 = NexProgressBar(0,
6, "j3");
    NexText velocidade_val   = NexText(0,
12, "t6");
/*int*/float  consumo_value = 0;
int     consumo_bar_value = 0;
    NexProgressBar j4 = NexProgressBar(0,
7, "j4");
    NexText consumo_val   = NexText(0, 14,
"t8");
#define OBD_CMD_RETRIES 3
boolean rpm_valid;
boolean obd_error_flag;
boolean rpm_error_flag;
boolean rpm_retries;
unsigned int rpm;
#define RPM_CMD_RETRIES 5
boolean consumo_valid;
boolean consumo_error_flag;
boolean consumo_retries;
unsigned int consumo;
#define CONSUMO_CMD_RETRIES 5
String mensagem_para_enviar = "";
String resposta_recebida_elm327 ="";
unsigned long espera = 0,espera2 = 0;
const int LED1=7, LED2=8;
char caracter_arduino, caracter_bluetooth;
boolean temperatura_valid;
boolean temp_error_flag;
boolean temp_retries;
int temp,temp_to_disp;
#define TEMP_CMD_RETRIES 5
boolean velocidade_valid;
boolean velocidade_error_flag;
boolean velocidade_retries;
unsigned int velocidade;
#define VELOCIDADE_CMD_RETRIES 5
boolean load_valid;
```

```
boolean load_error_flag;
boolean load_retries;
int load;
#define LOAD_CMD_RETRIES 5
boolean acelerador_error_flag;
boolean acelerador_retries;
unsigned int acelerador;
#define ACELERADOR_CMD_RETRIES 5
float nivel_combustivel;
boolean combustivel_error_flag;
boolean combustivel_retries;
unsigned int combustivel;
#define combustivel_CMD_RETRIES 5
int tentativas =0;
void setup() {
   Serial2.begin(9600); .
   nexInit();
   pinMode(led, OUTPUT);
   pinMode(pot, INPUT);
   digitalWrite(led, LOW);
   bluetooth_serial.begin(9600);
   Serial.begin(9600);
  obd_error_flag=false;
#ifndef DEBUG_SERIAL_ENABLE
  send_OBD_cmd("ATZ");
  delay(1000);
  Serial.print("RESPOSTA ATZ: ");
  Serial.println(resposta_recebida_elm327);
  resposta_recebida_elm327="";
  send_OBD_cmd("ATSP0");
  Serial.print("RESPOSTA ATSP0: ");
  Serial.println(resposta_recebida_elm327);
  resposta_recebida_elm327="";
#endif
 delay(2000);
 Serial.println("INICIANDO: ");
 delay(1000);}
void loop()
{mostra_total ="";
    while(tentativas<2)
    {temp_calc_teste();
     tentativas++;
       if(temperatura_valid)
         temperatura_display();
        if(t<12)t++;
        else t=0;}
     tentativas=0;
     while(tentativas<2)
     {  rpm_calc_teste();
       tentativas++;
     if(rpm_valid)
        rpm_display();
        if(y<12)y++;
        else y=0;}
     tentativas=0;
     while(tentativas<2)
     {  load_calc_teste();
       tentativas++;
       if(load_valid)
        carga_motor_display();
        if(c<12)c++;
        else c=0;   }
     tentativas=0;
     while(tentativas<2)
     {  velocidade_calc_teste();
       tentativas++;
        if(velocidade_valid)
        velocidade_display();
        if(v<12)v++;
        else v=0;  }
     tentativas=0;
     while(tentativas<3)
     {  nivel_comb_calc_teste();
       tentativas++;
        if(consumo_valid)
        consumo_display();
        if(cons<12)cons++;
        else cons=0;  }
     tentativas=0;
Serial.print(":Velocidade:");
Serial.print( velocidade);
Serial.print(":Cargamotor:");
Serial.print(load);
Serial.print(":%");
Serial.print(":Temperatura:");
Serial.print(temperatura_value);
Serial.print(":RPM:");
Serial.print(rpm);
Serial.print(":Nivelcombustivel:");
Serial.print(nivel_combustivel);
Serial.print(":Consumo:");
Serial.println(consumo);
 if (((millis() - espera) > 15) &&
(mensagem_para_enviar != ""))
  {
bluetooth_serial.print(mensagem_para_envia
r);
     Serial.print("mensagem para enviar: ");
     Serial.println(mensagem_para_enviar);
    mensagem_para_enviar = ""; }
  if (bluetooth_serial.available())
  { caracter_bluetooth =
bluetooth_serial.read();
   Serial.print(caracter_bluetooth);} }
```

Comandos na porta OBD

```
void send_OBD_cmd(char *obd_cmd)
{ char recvChar;
  boolean prompt;
  int retries;
   if (!(obd_error_flag))
   {prompt=false;
   retries=0;
   while((!prompt) &&
(retries<OBD_CMD_RETRIES))
    { bluetooth_serial.print(obd_cmd);
      bluetooth_serial.print("\r");
      Serial.print(obd_cmd);
      Serial.print("\r");
      Serial.println("Esperando Resposta...");
      while (bluetooth_serial.available() <= 0);
      while ((bluetooth_serial.available()>0)
&& (!prompt))
     {recvChar = bluetooth_serial.read();
       Serial.print(recvChar);
       resposta_recebida_elm327+=recvChar;
      if (recvChar==62) prompt=true;}
     retries=retries+1;
     delay(2000);}
   if (retries>=OBD_CMD_RETRIES) {
     obd_error_flag=true;
      digitalWrite(13,HIGH);}}}
```

Comando display

```
void rpm_display()
{                   rpm_value=rpm;
   memset(txt1, 0, sizeof(txt1));
   itoa(rpm_value, txt1, 10);
   rpm_bar_value = map(rpm_value, 0,5000,
0, 100);
   j0.setValue(rpm_bar_value);
   rpm_val.setText(txt1);}
void temperatura_display()
{   temperatura_value=temp;
   memset(txt1, 0, sizeof(txt1));
   itoa(temperatura_value, txt1, 10);
   temperatura_bar_value =
map(temperatura_value, 0, 255, 0, 100);

   j1.setValue(temperatura_bar_value);
   temperatura_val.setText(txt1);}
void carga_motor_display()
{                   carga_motor_value = load;
   memset(txt1, 0, sizeof(txt1));
   itoa(carga_motor_value, txt1, 10);
   carga_motor_bar_value =
map(carga_motor_value, 0, 100, 0, 100);
   j2.setValue(carga_motor_bar_value);
   carga_motor_val.setText(txt1);}
void velocidade_display()
{   j3.setValue(velocidade_bar_value);
   velocidade_val.setText(txt1);
   velocidade_value=velocidade;
   memset(txt1, 0, sizeof(txt1));
   itoa(velocidade_value, txt1, 10);
   velocidade_bar_value =
map(velocidade_value, 0, 255, 0, 100);}
void consumo_display()
{
                    j4.setValue(consumo_bar_value
);
   consumo_val.setText(txt1);
  // consumo_value=consumo;
    consumo_value=nivel_combustivel;
   memset(txt1, 0, sizeof(txt1));
   itoa(consumo_value, txt1, 10);
   consumo_bar_value =
map(consumo_value, 0, 1023, 0, 100);}
```

Transformando hex para dec

```
unsigned int hexToDec(String hexString)
{ unsigned int decValue = 0;
  int nextInt;
  for (int i = 0; i < hexString.length(); i++) {
    nextInt = int(hexString.charAt(i));
    if (nextInt >= 48 && nextInt <= 57)
nextInt = map(nextInt, 48, 57, 0, 9);
    if (nextInt >= 65 && nextInt <= 70)
nextInt = map(nextInt, 65, 70, 10, 15);
    if (nextInt >= 97 && nextInt <= 102)
nextInt = map(nextInt, 97, 102, 10, 15);
    nextInt = constrain(nextInt, 0, 15);
    decValue = (decValue * 16) + nextInt;}
  return decValue;}
```

Carga do motor

```
void load_calc_teste(){
   boolean prompt,valid;
   char recvChar;
   char bufin[12];
   int i;
   unsigned long time_Out = 0;
    time_Out = millis();
  if (!(obd_error_flag)){
     valid=false;
     prompt=false;
#ifndef
DEBUG_TESTE_CARGA_ENABLE
#ifdef DEBUG_SERIAL_ENABLE
       Serial.print("0104");
       Serial.print("\r");
       while (Serial.available() <= 0);
#else
      bluetooth_serial.print("0104");
      bluetooth_serial.print("\r");
      while ( bluetooth_serial.available() <=
0);
#endif
    i=0;
                  while((millis() - time_Out) <
100)
                {#ifdef
DEBUG_SERIAL_ENABLE
        while(Serial.available())
       { recvChar = Serial.read();
         Serial.println(recvChar);
         if(recvChar!=32)
bufin[i++]=recvChar;}
#else
       while (bluetooth_serial.available())
       {recvChar = bluetooth_serial.read();
         if(recvChar!=32)
bufin[i++]=recvChar;}
#endif}
#endif
#ifdef DEBUG_TESTE_CARGA_ENABLE
for (int i=0;i<6;i++)
{bufin[i]=teste_carga_buffer[c][i];}
    Serial.print("Posição i: ");
    Serial.println(i);
    Serial.print("Posição y: ");
    Serial.println(c);
#endif
      int flag_prot=0;
   if ((bufin[0]=='4') && (bufin[1]=='1') &&
(bufin[2]=='0') && (bufin[3]=='4'))
   {flag_prot=1;}
    if ((bufin[5]=='4') && (bufin[6]=='1') &&
(bufin[7]=='0') && (bufin[8]=='4'))
```

```
    {flag_prot=2;}
   if((flag_prot==1)||(flag_prot==2))
   {valid=true;} else {
    valid=false;}
   if (valid)
   {load_retries=0;
    load_error_flag=false;
   int DecimalDecode=0;
   if(flag_prot==1)
   { String loadHex(bufin[4]);
    String loadHex2(bufin[5]);
    String
loadHexTotal=loadHex+loadHex2;

DecimalDecode=hexToDec(loadHexTotal);}
   if(flag_prot==2)
   { String loadHex3(bufin[9]);
    String loadHex4(bufin[10]);
    String
loadHexTotal1=loadHex3+loadHex4;

DecimalDecode=hexToDec(loadHexTotal1);
}
load=round((float(DecimalDecode)/255)*10
0);
    load_valid=valid;}
   if (!valid){
    load_error_flag=true;
    load_retries+=1;
    load=0;
    if
(load_retries>=LOAD_CMD_RETRIES)
obd_error_flag=true;}
    if ((obd_error_flag==true)){load=0;
load_retries=0;}}}
```

Consumo

```
void consumo_calc_teste()
{   boolean prompt,valid;
   char recvChar;
   char bufin[15];
   int i;
                unsigned long time_Out = 0;
   time_Out = millis();
   obd_error_flag=false;
  if (!(obd_error_flag))
 {    valid=false;
      prompt=false;
#ifndef
DEBUG_TESTE_CONSUMO_ENABLE
#ifdef DEBUG_SERIAL_ENABLE
      Serial.print("015E");
      Serial.print("\r");
      while (Serial.available() <= 0);
#else
      bluetooth_serial.print("015E");
      bluetooth_serial.print("\r");
      while (bluetooth_serial.available() <=
0);
#endif
     i=0;
               while((millis() - time_Out) <
2000) //fica dentro do while por 500 ms
              { #ifdef
DEBUG_SERIAL_ENABLE
       while(Serial.available())
      { recvChar = Serial.read();
       Serial.println(recvChar);
       if(recvChar!=32)
bufin[i++]=recvChar;}
#else
      while (bluetooth_serial.available())
      { recvChar = bluetooth_serial.read();
       if(recvChar!=32)
bufin[i++]=recvChar;}
#endif}
#endif
#ifdef
DEBUG_TESTE_CONSUMO_ENABLE
for (int i=0;i<8;i++)
{bufin[i]=consumo_teste_buffer[cons][i];}
    Serial.print("Posição i: ");
    Serial.println(i);
    Serial.print("Posição cpns: ");
    Serial.println(cons);
#endif
      int flag_prot=0;
   if ((bufin[0]=='4') && (bufin[1]=='1') &&
(bufin[2]=='5') && (bufin[3]=='E'))
   {flag_prot=1;}
    if ((bufin[5]=='4') && (bufin[6]=='1') &&
(bufin[7]=='5') && (bufin[8]=='E'))
    {flag_prot=2;}
   if((flag_prot==1)||(flag_prot==2))
    { valid=true;}
    else
    {valid=false;}
   if (valid)
   {   consumo_retries=0;
      consumo_error_flag=false;
    if(flag_prot==1)
   {  for (i=4;i<8;i++)
     {if ((bufin[i]>='A') && (bufin[i]<='F'))
        {bufin[i]-=55;}
       if ((bufin[i]>='0') && (bufin[i]<='9'))
       {bufin[i]-=48;}
       consumo=(consumo << 4) | (bufin[i] &
0xf);}}
    if(flag_prot==2)
   {for (i=9;i<13;i++)
    {if ((bufin[i]>='A') && (bufin[i]<='F'))
       {bufin[i]-=55;}
       if ((bufin[i]>='0') && (bufin[i]<='9'))
       {bufin[i]-=48;}
       consumo=(consumo << 4) | (bufin[i] &
0xf);}}
      consumo = consumo /20;
      consumo_valid=valid;}}
    if (!valid)
    { consumo_error_flag=true;
     consumo_retries+=1;
     consumo=0;
     if
(consumo_retries>=CONSUMO_CMD_RE
TRIES) obd_error_flag=true;}
    if ((obd_error_flag==true)){consumo=0;
consumo_retries=0;}}
```

Nível de combustível

```
void nivel_comb_calc_teste()
{boolean prompt,valid;
  char recvChar;
  char bufin[12];
  int i;
  unsigned long time_Out = 0;
   time_Out = millis();
 if (!(obd_error_flag))
 {  valid=false;
   prompt=false;
#ifndef
DEBUG_TESTE_TEMPERATURA_ENAB
LE
#ifdef DEBUG_SERIAL_ENABLE
      Serial.print("012F");
      Serial.print("\r");
      while (Serial.available() <= 0);
#else
     bluetooth_serial.print("012F");
     bluetooth_serial.print("\r");
     while ( bluetooth_serial.available() <=
0);
#endif
  i=0;
  while((millis() - time_Out) < 100)
  { #ifdef DEBUG_SERIAL_ENABLE
     while(Serial.available())
      { recvChar = Serial.read();
       Serial.println(recvChar);
       if(recvChar!=32)
bufin[i++]=recvChar;}
#else
     while (bluetooth_serial.available())
      {recvChar = bluetooth_serial.read();
       if(recvChar!=32)
bufin[i++]=recvChar; // diferente espaço   }
#endif}
#endif
#ifdef
DEBUG_TESTE_TEMPERATURA_ENAB
LE
for (int i=0;i<6;i++)
{bufin[i]=teste_temperatura_buffer[t][i];}
    Serial.print("Posição i: ");
    Serial.println(i);
    Serial.print("Posição y: ");
    Serial.println(t);
#endif
    int flag_prot=0;
   if ((bufin[0]=='4') && (bufin[1]=='1') &&
(bufin[2]=='2') && (bufin[3]=='F'))
   {flag_prot=1;}
    if ((bufin[5]=='4') && (bufin[6]=='1') &&
(bufin[7]=='2') && (bufin[8]=='F'))
    {flag_prot=2;}
   if((flag_prot==1)||(flag_prot==2))
                {valid=true;   }
                else
                {valid=false;}
   if (valid)
              {   temp_retries=0;
    temp_error_flag=false;
   if(flag_prot==1)
   { String nivel_combHex(bufin[4]);
                                String
nivel_combHex2(bufin[5]);
     String nivel_combHexTotal =
nivel_combHex+nivel_combHex2;

nivel_combustivel=hexToDec(nivel_combH
exTotal);}
   if(flag_prot==2)
   { String nivel_combHex3(bufin[9]);
     String nivel_combHex4(bufin[10]);
     String
nivel_combHexTotal1=nivel_combHex3+ni
vel_combHex4;

nivel_combustivel=hexToDec(nivel_combH
exTotal1);}
                nivel_combustivel
=(nivel_combustivel*100)/255;
   consumo_valid=valid;}
   if (!valid)
              {   temp_error_flag=true;
    temp_retries+=1;
    temp=0;
    if
(temp_retries>=TEMP_CMD_RETRIES)
obd_error_flag=true;}
    if ((obd_error_flag==true)){temp=0;
temp_retries=0;}}}
```

Rotações

```
void rpm_calc_teste()
{boolean prompt,valid;
   char recvChar;
   char bufin[15];
   int i;
              unsigned long time_Out = 0;
   time_Out = millis();
   obd_error_flag=false;
  if (!(obd_error_flag))
 {    valid=false;
      prompt=false;
#ifndef DEBUG_TESTE_ENABLE
#ifdef DEBUG_SERIAL_ENABLE
       Serial.print("010C");
       Serial.print("\r");
       while ( Serial.available() <= 0);
#else
      bluetooth_serial.print("010C");
      bluetooth_serial.print("\r");
      while ( bluetooth_serial.available() <=
0);
#endif
     i=0;
              while((millis() - time_Out) <
500)
```

```
                {#ifdef
DEBUG_SERIAL_ENABLE
      while(Serial.available())
       { recvChar = Serial.read();
        Serial.println(recvChar);
        if(recvChar!=32)
bufin[i++]=recvChar;}
#else
     while (bluetooth_serial.available())
      {recvChar = bluetooth_serial.read();
       if(recvChar!=32)
bufin[i++]=recvChar;}
#endif}
 #endif
#ifdef DEBUG_TESTE_ENABLE
for (int i=0;i<8;i++)
{bufin[i]=teste_buffer[y][i];}
    Serial.print("Posição i: ");
    Serial.println(i);
    Serial.print("Posição y: ");
    Serial.println(y);
#endif
       int flag_prot=0;
   if ((bufin[0]=='4') && (bufin[1]=='1') &&
(bufin[2]=='0') && (bufin[3]=='C'))
   {flag_prot=1;}
    if ((bufin[5]=='4') && (bufin[6]=='1') &&
(bufin[7]=='0') && (bufin[8]=='C'))
    {flag_prot=2;}
if((flag_prot==1)||(flag_prot==2))
     {valid=true;}
     else
     {valid=false;}
   if (valid)
   {   rpm_retries=0;
      rpm_error_flag=false;
      rpm=0;
   if(flag_prot==2)
   { for (i=9;i<13;i++)
    {if ((bufin[i]>='A') && (bufin[i]<='F'))
       {bufin[i]-=55;}
       if ((bufin[i]>='0') && (bufin[i]<='9'))
       {bufin[i]-=48;}
       rpm=(rpm << 4) | (bufin[i] & 0xf);}}
    if(flag_prot==1)
   { for (i=4;i<8;i++)
    {if ((bufin[i]>='A') && (bufin[i]<='F'))
       {bufin[i]-=55;}
       if ((bufin[i]>='0') && (bufin[i]<='9'))
       {bufin[i]-=48;}
       rpm=(rpm << 4) | (bufin[i] & 0xf);}}
      rpm=rpm >> 2;
      rpm_valid=valid;
      mostra_rpm="RPM :"+rpm;}}
    if (!valid)
    { rpm_error_flag=true;
      rpm_retries+=1;
      rpm=0;
      if
(rpm_retries>=RPM_CMD_RETRIES)
obd_error_flag=true;}
     if ((obd_error_flag==true)){rpm=0;
rpm_retries=0;}}
```

Temperatura do liquido

```
void temp_calc_teste()
{boolean prompt,valid;
  char recvChar;
  char bufin[12];
  int i;
  unsigned long time_Out = 0;
   time_Out = millis();
 if (!(obd_error_flag))
 {valid=false;
    prompt=false;
#ifndef
DEBUG_TESTE_TEMPERATURA_ENAB
LE
#ifdef DEBUG_SERIAL_ENABLE
      Serial.print("0105");
      Serial.print("\r");
      while (Serial.available() <= 0);
#else
     bluetooth_serial.print("0105");
     bluetooth_serial.print("\r");
     Serial.println();
     while ( bluetooth_serial.available() <=
0);
#endif
  i=0;
  while((millis() - time_Out) < 100)
  { #ifdef DEBUG_SERIAL_ENABLE
     while(Serial.available())
      { recvChar = Serial.read();
       Serial.println(recvChar);
       if(recvChar!=32)
bufin[i++]=recvChar;}
#else
     while (bluetooth_serial.available())
      {recvChar = bluetooth_serial.read();
       if(recvChar!=32)
bufin[i++]=recvChar;}
#endif}
#endif
#ifdef
DEBUG_TESTE_TEMPERATURA_ENAB
LE
for (int i=0;i<6;i++)
{bufin[i]=teste_temperatura_buffer[t][i];}
    Serial.print("Posição i: ");
    Serial.println(i);
    Serial.print("Posição y: ");
    Serial.println(t);
#endif
    int flag_prot=0;
   if ((bufin[0]=='4') && (bufin[1]=='1') &&
(bufin[2]=='0') && (bufin[3]=='5'))
   {flag_prot=1;}
    if ((bufin[5]=='4') && (bufin[6]=='1') &&
(bufin[7]=='0') && (bufin[8]=='5'))
    {flag_prot=2;}
   if((flag_prot==1)||(flag_prot==2))
                {valid=true;}
                else
                {valid=false;}
   if (valid)
                {   temp_retries=0;
    temp_error_flag=false;
   if(flag_prot==1)
   { String tempHex(bufin[4]);
                                String
tempHex2(bufin[5]);
     String
tempHexTotal=tempHex+tempHex2;
      temp=hexToDec(tempHexTotal);}
   if(flag_prot==2)
   { String tempHex3(bufin[9]);
     String tempHex4(bufin[10]);
     String
tempHexTotal1=tempHex3+tempHex4;
      temp=hexToDec(tempHexTotal1);}
                                temp-=40;
   temperatura_valid=valid;}
   if (!valid)
                {   temp_error_flag=true;
    temp_retries+=1;
    temp=0;
    if
(temp_retries>=TEMP_CMD_RETRIES)
obd_error_flag=true;}
     if ((obd_error_flag==true)){temp=0;
temp_retries=0;}}}
```

Velocidade

```
void velocidade_calc_teste(){
  boolean prompt,valid;
  char recvChar;
  char bufin[12];
  int i;
   unsigned long time_Out = 0;
   time_Out = millis();//conta tempo
 if (!(obd_error_flag)){
    valid=false;
    prompt=false;
#ifndef
DEBUG_TESTE_VELOCIDADE_ENABL
E
#ifdef DEBUG_SERIAL_ENABLE
      Serial.print("010D");
      Serial.print("\r");
      while (Serial.available() <= 0);
#else
      bluetooth_serial.print("010D");
      bluetooth_serial.print("\r");
      while (bluetooth_serial.available() <=
0);
#endif
    i=0;
                while((millis() - time_Out) <
100)
               { #ifdef
DEBUG_SERIAL_ENABLE
       while(Serial.available())
      { recvChar = Serial.read();
       Serial.println(recvChar);
       if(recvChar!=32)
bufin[i++]=recvChar;}
#else
       while (bluetooth_serial.available())
       {recvChar = bluetooth_serial.read();
        if(recvChar!=32)
bufin[i++]=recvChar;}
#endif}
#endif
#ifdef
DEBUG_TESTE_VELOCIDADE_ENABL
E
for (int i=0;i<6;i++)
{bufin[i]=teste_velocidade_buffer[v][i];}
    Serial.print("Posição i: ");
    Serial.println(i);
    Serial.print("Posição v: ");
    Serial.println(v);
#endif
     int flag_prot=0;
   if ((bufin[0]=='4') && (bufin[1]=='1') &&
(bufin[2]=='0') && (bufin[3]=='D'))
   {flag_prot=1;}
    if ((bufin[5]=='4') && (bufin[6]=='1') &&
(bufin[7]=='0') && (bufin[8]=='D'))
    {flag_prot=2;}
    if((flag_prot==1)||(flag_prot==2))
    {valid=true;}
    else
    {valid=false;}
   if (valid){velocidade_retries=0;
    velocidade_error_flag=false;
      if(flag_prot==1)
   {   String velocidadeHex(bufin[4]);
       String velocidadeHex2(bufin[5]);
       String
velocidadeHexTotal=velocidadeHex+velocid
adeHex2;

velocidade=hexToDec(velocidadeHexTotal);
}
   if(flag_prot==2)
   { String velocidadeHex3(bufin[9]);
     String velocidadeHex4(bufin[10]);
     String
velocidadeHexTotal1=velocidadeHex3+velo
cidadeHex4;

velocidade=hexToDec(velocidadeHexTotal1
);}
      velocidade_valid=valid;}
   if (!valid){velocidade_error_flag=true;
    velocidade_retries+=1;
    velocidade=0;
    if
(velocidade_retries>=VELOCIDADE_CMD
_RETRIES) obd_error_flag=true;}
     if
((obd_error_flag==true)){velocidade=0;
velocidade_retries=0;}}}
```