UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
CURSO DE ENGENHARIA ELÉTRICA
CAMPUS CARAÚBAS

BEMIELISON GLETSON DA SILVA BEZERRA

DESENVOLVIMENTO DE UMA UNIDADE DE COMANDO ELETRÔNICO COM INTEGRAÇÃO À REDE CAN

CARAÚBAS-RN
2016

BEMIELISON GLETSON DA SILVA BEZERRA

DESENVOLVIMENTO DE UMA UNIDADE DE COMANDO ELETRÔNICO COM INTEGRAÇÃO À REDE CAN

Monografia apresentada a Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Engenheiro Eletricista.

Orientador: Prof. M.Sc. José Ailton L. Barboza Júnior – UFERSA.

CARAÚBAS-RN
2016

Dados Internacionais de Catalogação na Publicação (CIP)
Biblioteca Central Orlando Teixeira (BCOT)
Setor de Informação e Referência (SIR)

B574d Bezerra, Bemielison Gletson da Silva.
Desenvolvimento de uma unidade de comando eletrônico com integração à rede CAN / Bemielison Gletson da Silva Bezerra. - 2016.
70 f. : il.

Orientador: José Ailton Leão Barboza Júnior.
Monografia (graduação) - Universidade Federal Rural do Semi-árido, Curso de Engenharia Elétrica, 2016.

1. Unidade de comando eletrônico. 2. Rede CAN. 3. Arduino. 4. OBD-II. I. Júnior, José Ailton Leão Barboza , orient. II. Título.

O serviço de Geração Automática de Ficha Catalográfica para Trabalhos de Conclusão de Curso (TCC´s) foi desenvolvido pelo Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (USP) e gentilmente cedido para o Sistema de Bibliotecas da Universidade Federal Rural do Semi-Árido (SISBI-UFERSA), sendo customizado pela Superintendência de Tecnologia da Informação e Comunicação (SUTIC) sob orientação dos bibliotecários da instituição para ser adaptado às necessidades dos alunos dos Cursos de Graduação e Programas de Pós-Graduação da Universidade.

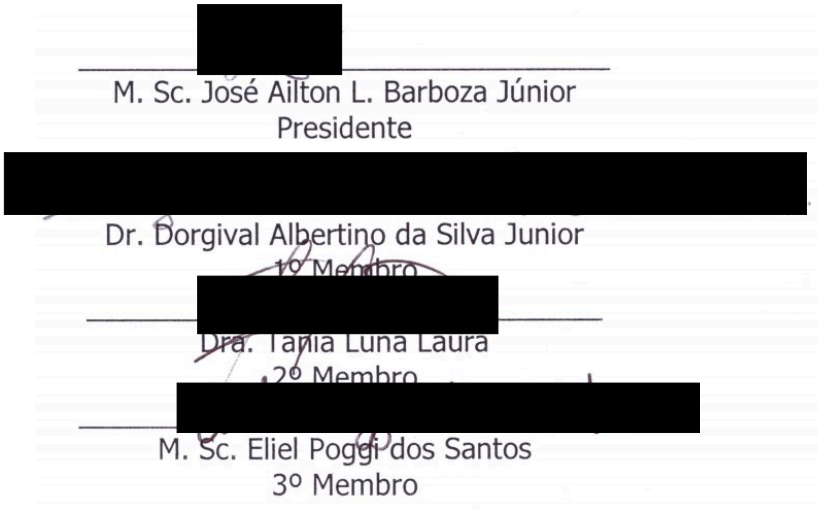
BEMIELISON GLETSON DA SILVA BEZERRA

DESENVOLVIMENTO DE UMA UNIDADE DE COMANDO ELETRÔNICO COM INTEGRAÇÃO À REDE CAN

Monografia apresentada a Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Engenheiro Eletricista.

Defendida em: 30 / 05 / 2016.

BANCA EXAMINADORA



AGRADECIMENTOS

Agradeço a Deus pelo fôlego de vida;

Agradeço a minha família por todo carinho e dedicação;

Agradeço ao Orientador pela disponibilidade e motivação;

Agradeço a todos os docentes que verdadeiramente se propuseram a ensinar;

Agradeço as minhas amigas Patrícia Rodrigues e Plycia Fernandes pelo incentivo;

Agradeço aos demais companheiros de estudo pela sinergia rumo à graduação;

Aos grandes nomes da engenharia elétrica que dedicaram suas vidas à busca pelo improvável, que mesmo sendo duramente desacreditados por tantos, mantiveram a fé e atingiram o objetivo maior que foi presentear a humanidade com tantas descobertas grandiosas.
A estes gigantes,

DEDICO.

“Isso é conectividade. Se você conectar o barramento à internet, o céu é o limite.”

Jomar Napoleão, SAE Brasil

RESUMO

Cada vez mais sistemas elétricos e eletrônicos têm sido adicionados aos veículos, sejam itens de conforto, gerenciamento do motor ou segurança. Estes sistemas requerem a utilização de diversos sensores e atuadores, implicando em grandes quantidades de fios e conexões que não apenas elevam o peso e o custo do veículo, mas torna complexa a confecção dos chicotes de ligação. Unidades de comando eletrônico (UCE) foram então embarcadas aos veículos reduzindo peso, custos e aumentando a velocidade de comunicação entre os sistemas. Este trabalho busca o desenvolvimento de uma unidade de comando eletrônico capaz de estabelecer comunicação com a rede de bordo CAN (*Controller Area Network)* do veículo, obter informações sobre os sistemas embarcados e realizar ações de controle de atuadores baseadas nas informações recebidas. Justifica-se pela relativa importância em inserir a Universidade Federal Rural do Semi-Árido no contexto de pesquisas relacionadas à área automotiva disponibilizando à comunidade acadêmica material de apoio ao desenvolvimento de soluções a serem embarcadas em veículos automotores. Para alcançar estes objetivos realizou-se (i) revisão da literatura acerca da evolução dos sistemas automotivos com foco na eletrônica embarcada, principais sistemas controlados eletronicamente, e quais componentes integram uma unidade de comando eletrônico; (ii) especificação da plataforma para gerenciamento das operações de entrada, processamento e saída; (iii) levantamento bibliográfico sobre o protocolo de comunicação CAN, padrão em veículos automotores; (iv) identificação dos parâmetros de requisição de informações através dos protocolos de diagnósticos OBD-II (*On Board Diagnostics*); (v) implementação de *firmware* para controle do protótipo de unidade de comando eletrônico, proposta deste trabalho; (vi) por fim, validação do protótipo através da realização de testes em campo utilizando veículo automotor. Portanto, o desenvolvimento deste trabalho servirá de base para pesquisas futuras relacionadas, ao disponibilizar material didático de apoio, seja por meio de texto ou do protótipo da unidade desenvolvido como modelo.

Palavras-Chaves: Unidade de Comando Eletrônico (UCE), Rede CAN, Arduino, OBD-II.

ABSTRACT

More and more electrical and electronic systems have been added to vehicles, are items of comfort, engine management or security. These systems require the use of several sensors and actuators, resulting in large numbers of wires and connections which not only increase the weight and cost of the vehicle but complicates the making of the connection harnesses. Electronic control units (ECU) are then shipped to reducing vehicle weight, cost and increasing the speed of communication between the systems. This work seeks to develop an electronic control unit able to communicate with the board CAN network (Controller Area Network) of the vehicle, information on embedded systems and perform actuator control actions based on the information received. Justified by the relative importance of inserting the Federal Rural University of Semi-Arid Region in the context of research related to the automotive industry by providing the academic community material support to the development of solutions to be embedded in automotive vehicles. To achieve these goals held (i) review of the literature on the evolution of automotive systems with a focus on embedded electronics, key electronically controlled systems and components which are part of a electronic control unit; (ii) platform specification for managing inbound operations, processing and output; (iii) literature on the CAN communication protocol, standard in motor vehicles; (iv) identification of information request parameters using the OBD-II diagnostic protocols (On Board Diagnostics); (v) implementation of firmware for prototype control electronic control unit, purpose of this work; (vi) finally, prototype validation by conducting field tests using a motor vehicle. Therefore, the development of this work will form the basis for related future research to provide educational material support, either through text or prototype unit developed as a model.

Keywords: Electronic Control Unit (ECU), CAN Network, Arduino, OBD-II.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

A/D	Analógico para Digital
ABS	*Anti-lock Breaking System*
ACK	*Acknowledgment error check*
ASR	*Anti-Slip Regulation*
AVR	Microcontrolador ATmel de arquitetura RISC
CAD	*Computer-Aided Design*
CAN	*Controller Area Network*
CRC	*Cyclical Redundacy Check*
D/A	Digital para Analógico
DIN	*Deutsche Industrie Norm*
DIY	*Do It Yourself*
EBD	*Eletronic Brake Distribution*
ECM	*Engine Control Module*
ESP	*Eletronic Stability Program*
GPL	*General Public License*
GPS	*Global Positioning System*
GSM	*Global System for Mobile Communications*
IDE	*Integrated Development Environment*
ISO	*International Organization for Standardization*
KBPS	Quilobyte por segundo
kPa	QuiloPascal
LCD	Display de Cristal Líquido
LGPL	*Lesser General Public License*
NMEA	*La National Marine Electronics Association*
OBD	*On Board Diagnostics*
OSI	*Open Systems Interconnection*
PID	*Parameter ID*
PWM	*Pulse Width Modulation*
RISC	*Reduced Instruction Set Computer*
RPM	Rotações por Minuto
SAE	*Society of Automotive Engineers*

SPI	*Serial Peripheral Interface*
TCS	*Traction Control System*
TCU	*Transmission Control Unit*
UART	*Universal Asynchronous Receiver-Transmitter*
UCE	Unidade de Comando Eletrônico
USB	*Universal Serial Bus*
L/h	Litros por hora
km/h	Quilômetros por hora

LISTA DE SÍMBOLOS

%	Porcentagem
ºC	Graus Celsius
V	Volts

SUMÁRIO

1 INTRODUÇÃO

Cada vez mais sistemas elétricos e eletrônicos têm sido adicionados aos veículos, sejam itens de conforto, gerenciamento do motor ou segurança. Estes sistemas requerem a utilização de diversos sensores e atuadores, implicando em grandes quantidades de fios e conexões que não apenas elevam o peso e o custo do veículo, mas torna complexa a confecção dos chicotes de ligação. Outro problema relacionado se refere à comunicação entre os sistemas a serem monitorados, onde a ausência de um protocolo de comunicação causa atraso nas respostas e consequentemente riscos à segurança (BOSCH, 2005).

Unidades de Controle Eletrônico (UCE) são sistemas computacionais interligados em rede (predominantemente redes CAN – *Controller Area Network*) e embarcados em veículos automotores. Uma UCE é projetada para fins específicos, desempenhando basicamente atividades de monitoramento de sensores e acionamento de atuadores presentes no veículo. Cada variável do veículo a ser controlada se concentra em uma UCE, divididas por área ou nós, a critério de cada fabricante. Os nós mais comuns são: de controle do vão do motor, painel, quadro de instrumentos, diagnóstico, direção elétrica, rádio, ar condicionado, alarme e travas das portas, sistema de freio (ABS, EBD, etc.) e outros que podem ser adicionados à rede, desde que seja obedecido o protocolo de comunicação padrão do veículo. O emprego de UCE possibilita redução significativa do peso, de cabos, terminais e conexões, totalizando cerca de 23% (BARBOSA, 2003).

O protocolo de comunicação serial predominante, mas não único, é o CAN. Projetado pela empresa BOSCH, o protocolo CAN tornou-se padrão mundial de comunicação entre as unidades de comando eletrônico do veículo. O protocolo de comunicação em veículos é dividido em três áreas de acordo com sua velocidade. São elas: comunicações entre centrais (125 kbit/s a 1 Mbit/s), conveniência e conforto interno, juntamente com comunicação móvel (50 kbit/s a 125 kbit/s).

1.1 Objetivos

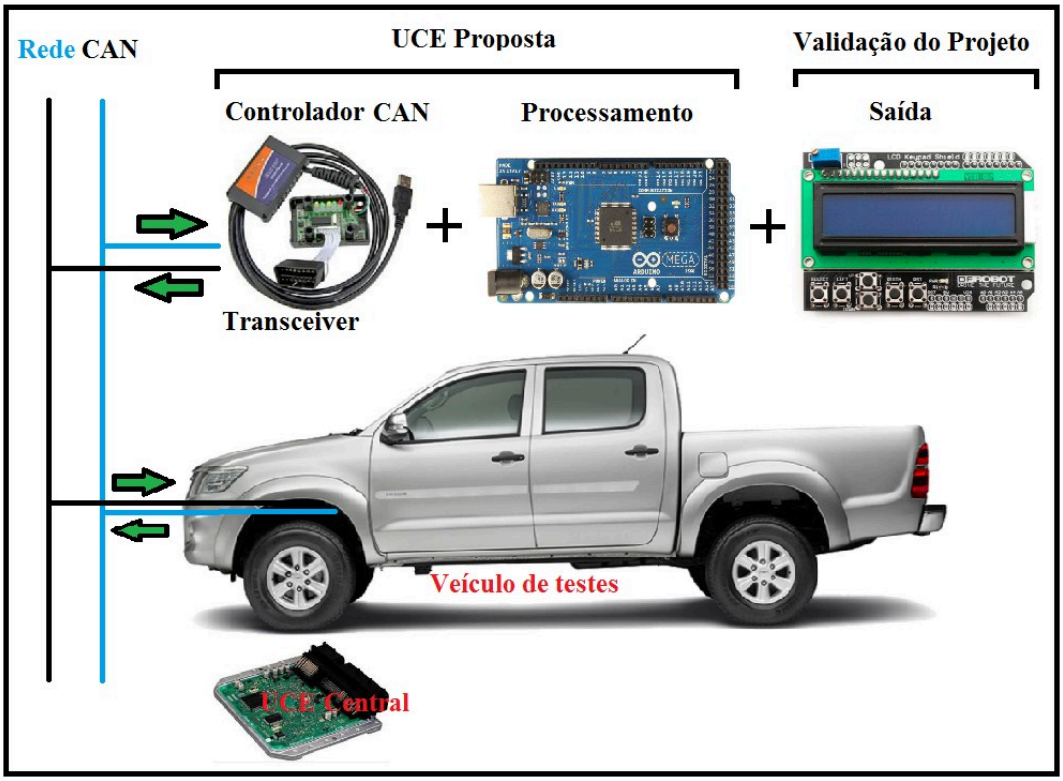
Frente à relevância do tema exposto, o presente trabalho tem como objetivo geral inserir a Universidade Federal Rural do Semi-Árido (UFERSA) neste cenário mundial de pesquisas relacionadas a tecnologias automotivas ao desenvolver um protótipo de unidade de controle eletrônico capaz de ser integrado à rede CAN, obter informações da rede e executar

ações (acionar atuadores) baseadas nas informações coletadas.

Os objetivos específicos são:

- Realizar levantamento bibliográfico acerca das unidades de comando eletrônico;
- Compreender o processo de comunicação entre a UCE e demais componentes da rede;
- Especificar os componentes necessários para o desenvolvimento de uma UCE;
- Construir o protótipo de UCE;
- Implementar *firmware* de controle da UCE a ser construída;
- Inserir interface de monitoramento da rede CAN à UCE construída;
- Realizar testes de comunicação entre o protótipo de UCE e a UCE principal.

Figura 01 – Diagrama do Sistema Proposto.



Fonte: Montagem autoria própria a partir de imagens coletadas do Google Imagens.

1.2 Justificativa

No mundo, o protocolo CAN tornou-se padrão de referência no que se refere à redes de comunicação automotiva. Requisitos de inúmeros subsistemas específicos do veículo automotivo, em termos de funcionamento, segurança, compatibilidade ambiental e comodidade, só podem ser satisfeitos através de conceitos de comando e regulação altamente desenvolvidos (BOSCH, 2005).

Sabendo que os veículos automotores estão cada vez mais modernos, seus recursos de conforto, segurança e entretenimento, e que o gerenciamento dos sensores e atuadores demandará a cada dia a utilização de mais UCE´s, dominar as técnicas de projeto e desenvolvimento dessas unidades torna-se imprescindível. Qualquer novo recurso eletrônico a ser inserido deverá desde as etapas de projeto garantir possibilidade de ser conectado à rede e esta obrigatoriedade exigirá conhecimentos sólidos do assunto.

O estudo de técnicas de projeto, construção e montagem, a vivência prática na resolução de problemas e propostas de soluções permitirá o desenvolvimento de novas soluções aplicáveis aos veículos automotivos e assim contribuirá para o progresso tecnológico do setor.

Diante do exposto, torna-se perceptível a tendência de integração à rede de quaisquer novos componentes eletrônicos ao veículo automotor, então o conhecimento relacionado ao desenvolvimento de UCE´s, incluindo sua integração à rede, mostra-se a base para implementação de qualquer novo sistema automotivo, seja para segurança ou conforto. Deste modo, tal conhecimento torna-se essencial para qualquer engenheiro da área, uma vez que possibilita a manipulação das variáveis do sistema, tais como: redução de custos, velocidade e robustez do sistema.

2 REFERENCIAL TEÓRICO

2.1 Unidade de comando eletrônico

No que se refere a uma UCE, esta se subdivide em condicionamento dos sinais de entrada, processamento lógico e saída. Entre os sistemas integrados aos veículos e controlados por UCE´s pode-se citar: o controle de tração (ASR) e sua extensão; o controle eletrônico de estabilidade (ESP) que verifica por meio de sensores se uma das rodas gira em falso, então a UCE informa via rede à UCE de gerenciamento do motor para que esta determine uma redução correspondente no torque de tração (BOSCH, 2005).

Figura 02 – UCE fabricação Bosch.

Fonte: www.bosch.com.

2.1.1 Histórico

Não se sabe precisamente a data da invenção do automóvel, porém há conhecimento de que o primeiro, cuja propulsão se dava a vapor, remonta à data de 1769 e somente em 1885 surge o primeiro motor de combustão interna. Ao longo de décadas muito mudou até se chegar aos modelos atuais e muitas das inovações se deram por meio de pesquisas iniciadas com foco nas pistas de corrida, em competições que testam não apenas pilotos, mas os limites das máquinas. Henry Ford, através de sua linha de montagem para fabricação em série do Ford T revolucionou, ultrapassando os 15 milhões de unidades fabricadas.

Enquanto a Europa e os Estados Unidos montavam suas máquinas, no Brasil esse

segmento ainda era um mundo desconhecido até a chegada das montadoras, salvo para aqueles de boa condição financeira e que viessem a realizar importação.

Figura 03 – Ford Violet, 1892, anterior ao Ford T.

Fonte: ANJOS, 2011.

Com o passar dos anos, o aperfeiçoamento dos veículos se deu em diversos aspectos, desde estético ao conforto e eficiência. Diversos novos sistemas foram embarcados, inclusive sistemas elétricos e eletrônicos, impulsionados pelos avanços tecnológicos na área de microeletrônica.

Um sistema de gerenciamento eletrônico para esses novos recursos de monitoramento e atuação tornou-se imprescindível sendo então aplicado o conceito de UCE. O termo sistema eletrônico embarcado tornou-se cada vez mais usual e se refere ao desenvolvimento de um sistema para realização uma ou mais funções específicas, na maioria das vezes em tempo real e assim como um computador pessoal capaz de controlar diversas necessidades dos usuários.

Unidades micro processadas centrais controlam módulos espalhados por todo veículo e estes módulos por sua vez realizam tarefas específicas de monitoramento e atuação. Como a maioria dos sensores e atuadores opera com sinais analógicos e o processador apenas com sinais digitais, há então conversores analógicos-digitais (A/D) para entrada de sinais e digitais-analógicos (D/A) para saída dos sinais já processados (ANJOS, 2011).

Entre os sistemas embarcados em veículos estão:

- *Anti-Lock Braking System* – ABS: Evita o travamento das rodas do veículo durante a frenagem;
- *Electronic Stability Control* – ESC/ESP: Controle de estabilidade do veículo;

- *Traction Control* – TCS: Controle de tração;
- *Engine Control Module* – ECM: Módulo de controle do motor;
- *Telematic Control Unit* – TCU: Módulo de controle telemático.

2.1.2 Sistemas *x-by-Wire*

É a nomenclatura dada a sistemas que substituem componentes mecânicos (botões, cabos de aço, etc.) por fios e cabos elétricos e assim acionar os mesmos sistemas já conhecidos (iluminação, acelerador, etc.). Para tal utilizam-se sensores cujos sinais são processados pelas centrais e posteriormente comandam os atuadores eletromecânicos de maior capacidade de monitoramento e ajustes. O termo "X" representa o sistema anteriormente mecânico que foi substituído por lógica eletrônica (ANJOS, 2011).

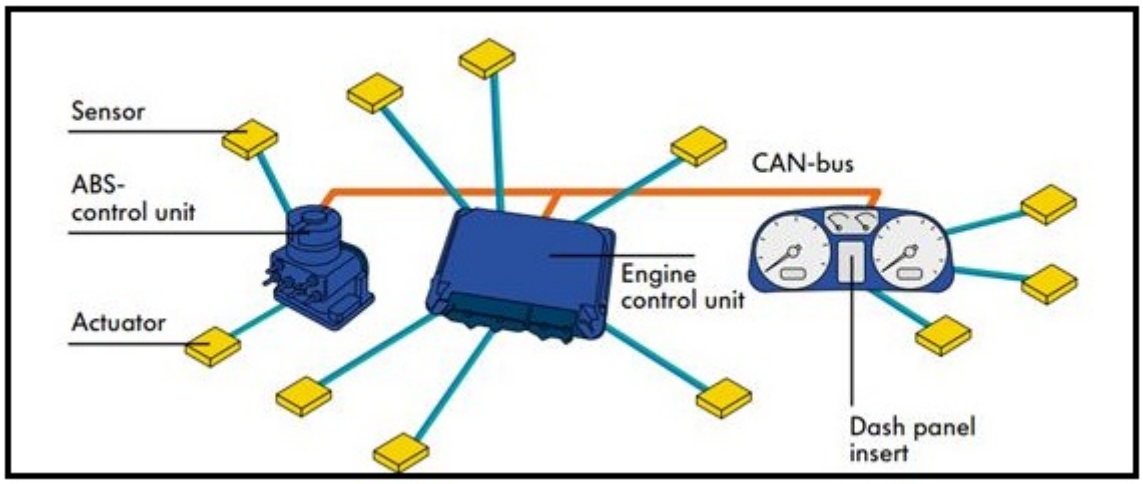
Alguns exemplos de aplicações seriam:

- *Gas-by-wire*: monitora a injeção e o acelerador eletrônico;
- *Power-by-wire*: Atua nos sistemas de partida;
- *Shift-by-wire*: Monitora as caixas de câmbio;
- *Brake-by-wire*: Controle de freios, estabilidade, e tração (ABS, EBD, TCS, e ESP);

É tendência atual que todos os sistemas mecânicos sejam substituídos por sistemas *x-by-wire* e as montadoras de veículos tem realizado seu papel e desenvolvido constantemente novos sistemas, pois a introdução desses sistemas agrega conforto, segurança, economia de combustível, dirigibilidade mais eficientes, entre outros benefícios que atraem o consumidor.

Os sistemas *x-by-wire* são conectados à rede de bordo e permitem a leitura de diversos parâmetros de diagnósticos.

Figura 04 – Unidade de controle conectada a sensores e atuadores.



Fonte: *Data Exchange On The CAN Bus I.*

2.2 Arduino

2.2.1 Filosofia

É uma plataforma de prototipagem eletrônica capaz de gerar interação entre *hardware* e o mundo físico através da implementação de códigos (algoritmos) inseridos em um microcontrolador (ATmel AVR, arquitetura RISC – *Reduced Instruction Set Computer*), em conjunto com sensores e atuadores que detectam e executam ações. O projeto iniciou em 2005, na cidade de Ivrea da Itália, com o intuito inicial de uso em projetos escolares acarretando o mínimo de custos possíveis em relação a outros sistemas de prototipagem da época. Já no ano de 2006 recebeu o prêmio de menção honrosa da *Prix Ars Electronica* na categoria Comunidades Digitais revelando assim seu sucesso (PLATAFORMA ARDUINO, 2016).

A escolha da plataforma Arduino para este trabalho ocorreu em virtude: (i) estreita relação entre os objetivos de ambos; (ii) alternativa de baixo custo de aquisição (iii) praticidade e facilidade de uso; (iv) disponibilidade de variados recursos de *hardware* e sensores/atuadores; (v) comunidade global de usuários, compartilhamento de bibliotecas e códigos exemplos. Deste modo, a plataforma Arduino é uma boa opção de uso para este projeto. A figura 05 ilustra a representação de uma das placas da plataforma Arduino.

Figura 05 – Arduino Uno R3 (vista superior e inferior).

Fonte: http://arduino.cc.

2.2.2 Histórico

Os *hardwares* da plataforma Arduino são *open-source* e regidos pela licença *Creative Commons*. O *software* (IDE JAVA) é liberado pela licença GPL e as bibliotecas C/C++ de microcontroladores estão liberadas pela licença LGPL. Deste modo, os projetos de *hardware* e *softwares* podem ser vistos e modificados, sendo possível inclusive propor melhorias e assim contribuir para o aperfeiçoamento da plataforma. A proposta é a disponibilização ao mundo de uma solução de custo reduzido e de manuseio simplificado para interação entre computadores e ambiente físico, reduzindo complicações e a necessidade de maiores conhecimentos de eletrônica e programação de computadores.

As placas Arduino podem captar o ambiente através de sensores conectados a entradas analógicas e/ou digitais e efetuar ações por meio de atuadores. É possível adquirir comercialmente qualquer *hardware* oficial Arduino pronto para uso, mas também há a opção de montagem manual, configurada pelo termo *DIY* (faça você mesmo), e para isso o *site* oficial [1] disponibiliza todo o projeto através de diagramas esquemáticos dos circuitos e arquivos CAD. É possível ainda visualizar referências para toda sintaxe da programação, bem como realizar o *download* de bibliotecas para as mais diversas aplicações envolvendo a plataforma (PLATAFORMA ARDUINO, 2016).

Além do *site* oficial, existem numerosos sítios com instruções passo a passo, imagens, vídeos e códigos de aplicações feitas e compartilhadas para uso por usuário de todas as partes do mundo. O Arduino é multiplataforma, podendo ser utilizado em sistemas operacionais Linux, Windows e Mac OS X. O *hardware* Arduino é programado utilizando sintaxe própria baseada em *Wiring*, mas essencialmente C/C++. O ambiente de desenvolvimento é baseado em *Processing*, mas é possível compilar o código utilizando outras ferramentas como

[1] http://arduino.cc/

Makefiles e/ou AVR *Studio*, ou em sistemas Linux por linha de comando sem a necessidade da instalação de JAVA.

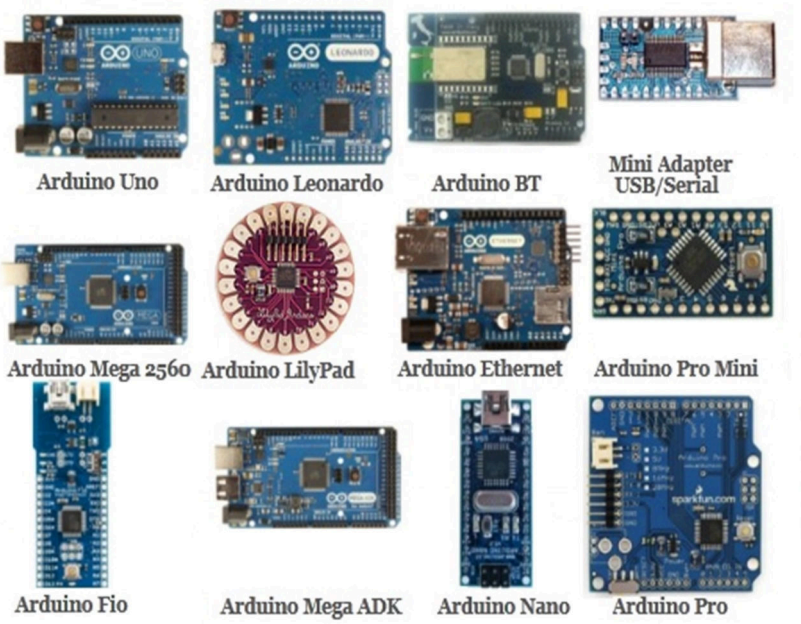
2.2.3 Descrição Técnica

A plataforma possui variados modelos de *hardwares* oficiais, que são empregados conforme a necessidade do usuário seja em número de entradas/saídas digitais e entradas analógicas, capacidade de memória de programa, número de saídas PWM, protocolo de comunicação, entre outros. Além das placas oficiais (ver figura 06), é possível encontrar diversas variações compatíveis com a plataforma e que se adequam as mais diversas necessidades de utilização, como por exemplo, a distribuição Freeduino, Brasuíno, e a Seeeduino, que proporcionam inúmeras possibilidades tantas quanto às versões oficiais do Arduino.

O *hardware* da plataforma Arduino foi desenvolvido inclusive para oferecer flexibilidade, sendo permitido expandir suas funcionalidades através de módulos de expansão que agregam recursos adicionais. Estes módulos recebem a nomenclatura de escudos (do inglês *shield*), e podem ser simplesmente conectados uns sobre os outros. Um exemplo da sua utilização é visto na figura 07, onde são agrupados ao *hardware* Arduino Uno 1 (um) *shield* de expansão para utilização de um leitor de cartão de memória e logo acima é conectado 1 (um) *Ethernet Shield* que permite a conexão do sistema a uma rede de dados. Este exemplo possibilita o monitoramento e controle das entradas e saídas digitais e/ou entradas analógicas remotamente, através de uma aplicação *Web* e que poderá ser acessada de qualquer lugar no planeta que disponha de acesso à internet. As informações podem ser ainda visualizadas em *Smartphone* ou *tablets*, e gerar sistemas de *SmartHouse*[2] (PLATAFORMA ARDUINO, 2016).

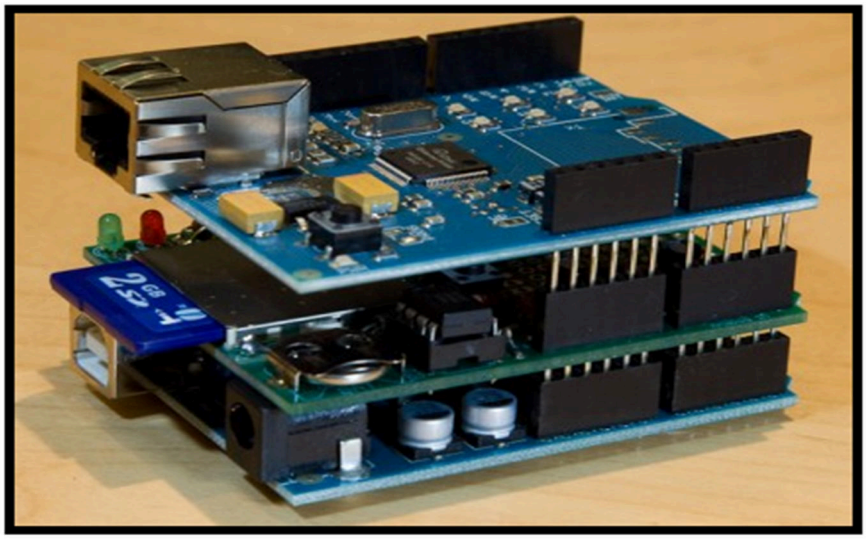
[2] Casa inteligente, conceito na qual os equipamentos e sistemas de uma casa possam ser controlados e monitorados local e/ou remotamente através de dispositivos computacionais como *smartphones*, *tablets* ou mesmo computadores *desktop* e *laptops*.

Figura 06 – Placas oficiais da plataforma Arduino.



Fonte: http://arduino.cc.

Figura 7 – Arduino Uno conectado ao *Ethernet Shield* e SD *Card Shield*.

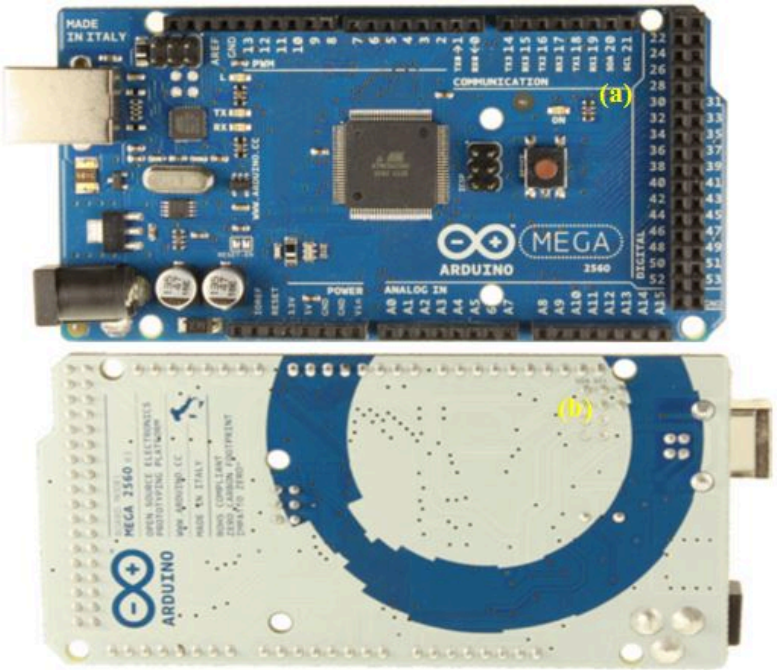


Fonte: http://hackaday.com.

2.2.4 Arduino Mega

A especificação do protótipo, objetivo deste trabalho, faz uso do hardware Arduino Mega 2560, ilustrado pela figura 08. Esta placa possui maior capacidade de memória comparada a do modelo Arduino Uno, tendo o custo de aquisição equivalente.

Figura 08 – Placa Arduino Mega 2560 - vista superior (a) e inferior (b).



Fonte: http://arduino.cc.

Além disso, a Arduino Mega 2560 apresenta grande quantidade de entradas/saídas digitais e entradas analógicas disponíveis que podem ser utilizadas para os mais diversos sistemas de sensores e atuadores conforme a necessidade no decorrer do uso e implementação de sistemas mais avançados, o que motivou a sua escolha para esse projeto.

Para melhor compreensão da capacidade de processamento são apresentados os dados técnicos da placa Arduino Mega 2560 na tabela 1.

Tabela 1 – Dados técnicos do hardware Arduino Mega 2560.

Arquivo *EAGLE* CAD:	Arduino-mega2560_R3-reference-design.zip
Diagrama esquemático:	Arduino-mega2560_R3-schematic.pdf
Mapa de pinagem:	PinMap2560 page
Microcontrolador:	ATmega2560
Tensão de operação:	5 Volts
Tensão de entrada (recomendável):	7 – 12 Volts
Tensão limite de entrada (extremos):	6-20 Volts
Pinos digitais (I/O):	54 no total (15 são também saídas PWM)
Pinos de entrada analógica:	16
Corrente DC por pino I/O:	40mA
Corrente DC para o pino de 3,3 Volts:	50mA
Memória Flash:	256KB dos quais 8KB são usados pelo bootloader[3]
SRAM:	8KB
EEPROM:	4KB
Clock Speed:	16MHz

Fonte: http://arduino.cc.

[3] Programa responsável por realizar o gerenciamento de inicialização do sistema.

O Arduino Mega pode ser alimentado via cabo USB (o mesmo utilizado para transferência de códigos) ou mesmo por fonte externa, seguindo a recomendação para faixa de 7,0 a 12,0 volts em corrente contínua. A placa dispõe ainda de conectores para encaixe dos condutores, e permitem fácil conexão e retirada dos condutores do circuito durante a realização das aplicações. Tais características facilitam na execução dos circuitos durante os testes de funcionamento, evidenciando a aplicabilidade didática da plataforma. A especificação dos pinos presentes na placa Arduino MEGA 2560 é apresentada na tabela 2.

Tabela 2 – Especificação dos pinos da placa Arduino Mega 2560.

Pinos	**Descrição**
Vin:	Pino de entrada de energia quando se está utilizando fonte de energia externa, como uma bateria de 9 volts, por exemplo.
5V:	Apresenta uma tensão de saída de 5V regulada, porém não é aconselhável a utilização de entrada de energia através deste pino, pois isto ignora o regulador e pode danificar a placa.
3V3:	Apresenta uma tensão de saída de 3,3V regulada, porém também não é aconselhável a utilização de entrada de energia através deste pino, pois isto ignora o regulador e pode danificar a placa.
GND:	Pinos de terra
TX:	Os pinos (1, 18, 16, 14) são utilizados como TX para transmitir dados seriais, porém apenas o pino “1” é ligado ao conversor USB pata TTL.
RX:	Os pinos (2, 19, 17, 15) são utilizados como RX para transmitir dados seriais, porém apenas o pino “2” é ligado ao conversor USB pata TTL.
PWM (Modulação por Largura de Pulso):	Estão disponíveis nos pinos de 2 a 13 e de 44 a 46, totalizando 15 saídas PWM.
SPI (*Serial Peripheral Interface Bus*):	Pino 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS). Dão suporte à comunicação SPI através da biblioteca ICSP.
TWI ou I²C (*Interface* de dois fios):	Pino 20 (SDA), 21 (SCL). Dão suporte à comunicação TWI através da biblioteca *Wire*.
Entradas analógicas:	A placa Arduino Mega possui 16 entradas analógicas numeradas de A0 a A15 e cada uma possui resolução de 10 bits, totalizando 1024 valores diferentes para uma escala entre zero e cinco volts.
Entradas e saídas digitais:	São numerados de zero à 53 e disponibilizam tensão de 5 volts quando em nível lógico alto (*High*), ou 0 volts quando em nível lógico baixo (*Low*).

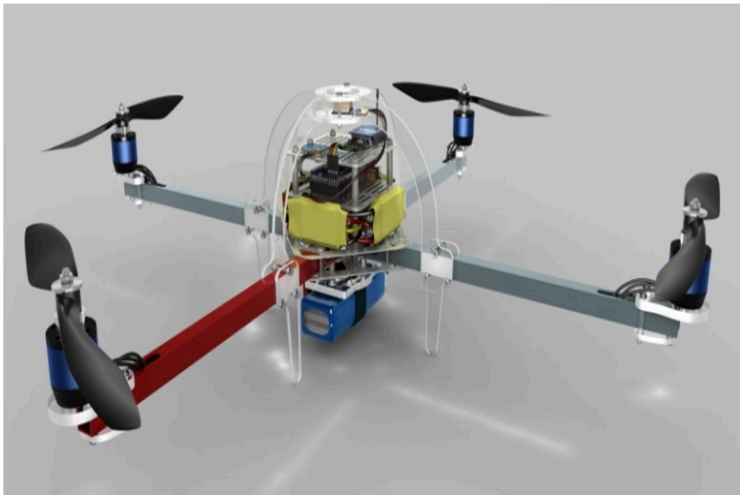
Fonte: http://arduino.cc

2.2.5 Exemplos de Aplicações com Arduino

2.2.5.1 *Arducopter*

O *ArduCopter* é uma plataforma para helicópteros multirrotores na qual o *hardware* de controle padrão é baseado no mega *ArduPilot* desenvolvido pela comunidade *Drones* DIY e comercializado pela empresa asiática *jDrones*, ilustrado na figura 09.

Figura 09 – *ArduCopter*, VANT quadrirrotor utilizando plataforma Arduino.



Fonte: http://diydrones.com.

O produto resultou em um quadrirrotor de fácil controle de manobras e capaz de ser controlado remotamente ou de modo autônomo incluindo planejamento de missões e telemetria que podem ser visualizados em uma estação terrestre. Possui sensores como magnetômetro e GPS, permitindo que o quadrirrotor retorne ao local de origem por si só (PROJETO ARDUCOPTER, 2016).

2.2.5.2 *Mobot BTCar*

O projeto *Mobot BTCar* faz uso da plataforma Arduino para modificar o modo de controle de um carrinho elétrico de brinquedo, ver figura 10. Antigamente, era controlado por rádio frequência, agora utiliza o protocolo de comunicação *Bluetooth* como meio de controle. Por meio de um *smartphone* ou *tablet*, com o sistema operacional Google Android[4] e o aplicativo *Mobot BTCar* embarcado, comandos podem ser enviados e executados no carrinho de brinquedo. É possível realizar ações como: alternar entre faróis (alto ou baixo), ligar lanternas traseiras, luz de marcha ré, movimentar o veículo para frente ou para trás, para direita ou esquerda. O sistema possibilita ainda o controle do veículo através do sensor acelerômetro do dispositivo móvel, onde ao inclinar um ou mais eixos em relação ao centro de gravidade do sensor este habilita a execução de ações em forma de movimento (MOBOT CAR, 2016).

[4] Plataforma completa para dispositivos móveis, possuindo *Kernel* GNU Linux, *Middleware* e aplicações, SDK e APIs para desenvolvimento na plataforma Android com o uso da linguagem JAVA.

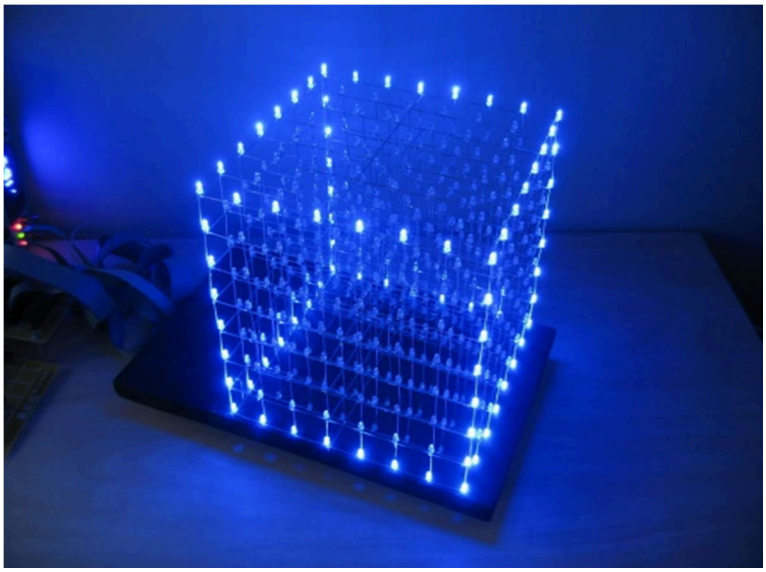
Figura 10 – Projeto *Mobot BTCar*.

Fonte: http://www.mobot.es.

2.2.5.3 *Led Cube* 8x8x8

O projeto *Led Cube* 8x8x8, também é baseado na utilização da plataforma Arduino, onde é possível exibir imagens em formato tridimensional. Os efeitos e formas apresentadas são pré-configuradas no microcontrolador e podem ser ativadas manualmente por um botão seletor ou dinamicamente pelo ritmo do som no ambiente. Um exemplo de configuração é visualizado na figura 11 (LED CUBE, 2016).

Figura 11 – Cubo de Led, matriz 8x8x8.



Fonte: http://www.instructables.com/id/Led-Cube-8x8x8/.

2.3 Rede CAN

2.3.1 Descrição da rede

A rede automotiva de dados CAN é padronizada mundialmente pela resolução ISO 11898 gerada pela *International Society of Organization*, e foi proposta por Robert Bosch em 1980 para interconexão de componentes de controle de veículos e é regulamentada pela *Society of Automotive Enginneers*, utilizando como referência a definição de camadas das redes de dados ISO/OSI, sendo seus níveis apenas o físico e de enlace (BARBOSA, 2003).

As redes CAN possuem excelentes características de confiabilidade em ambientes ruidosos e assim também vem sendo implementadas em aplicações industriais. Utiliza um barramento composto por um par de cabos onde ambos transmitem a mesma informação, em redundância, e mesmo que um dos cabos se rompa a informação não será prejudicada. A rede trabalha no princípio "multi-*master*", onde várias UCE´s de mesmo nível de prioridade são interconectadas ao mesmo barramento e assim, como vantagem, a falha de uma das unidades não implica em prejuízo de comunicação para as demais (BOSCH, 2005).

A rede CAN tornou-se a resposta para o gargalo anteriormente enfrentado pelas montadoras, uma vez que dia após dia multiplicava-se o número de sistemas elétricos e eletrônicos a serem gerenciados nos veículos acarretando sobrepeso de condutores, elevado custo de fabricação, dificuldade de manutenção e adequação de equipamentos em projetos futuros, entre outros. A rede é composta por apenas um único par de condutores que transporta a informação de modo multiplexado e em redundância para cada elemento do sistema. A informação é passada a todas as unidades, porém apenas a unidade correspondente à solicitação a processa e isto é realizado através do bit de prioridade (0 – Dominante, 1 - Recessivo) (MARQUES, 2004).

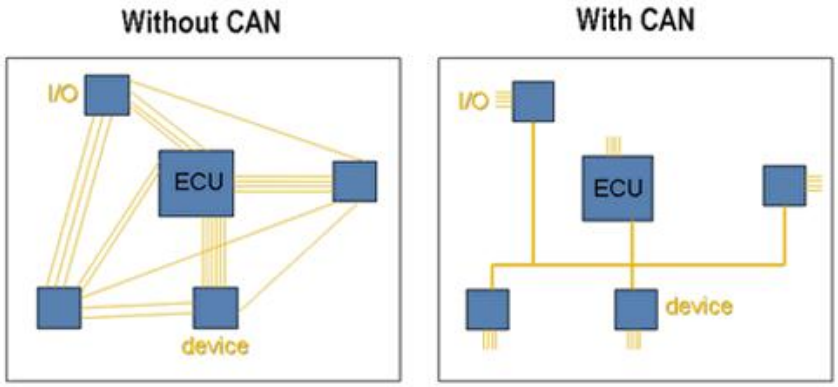
A rede CAN proporcionou grande flexibilidade aos projetos automotivos, onde qualquer novo componente (sensor ou atuador) a ser integrado ao projeto necessitará, basicamente, de uma retificação de *firmware* na UCE central e conexão à rede por meio de uma UCE específica. Sistemas que anteriormente eram totalmente mecânicos, a exemplo do sistema de pedal acelerador, estão sendo substituídos por sistemas eletromecânicos controlados pela rede de bordo, onde a variação do acionamento do pedal provoca, através da rede, o acionamento de um motor elétrico que por sua vez varia a aceleração do veículo. Outra importante característica se refere ao método de multiplexação para transferência de informações, permitindo que as centrais conversem entre si, em tempo real, sem a necessidade

de conexão *point-to-point* (cada central conectada a todas as outras por meios distintos) (MARQUES, 2004).

As principais características do protocolo CAN estão alicerçadas, segundo Sakai (2008), nos seguintes pilares:

- Rede multimestre, onde todos os nós tem permissão para utilizar o barramento, desde que esteja livre;
- Comunicação até 1 Mbps;
- Quadro (*frame*) com até 8 bytes de informação;
- Inserção, remoção ou mudança de dispositivos no barramento.

Figura 12 – Comparativo entre sistemas sem rede CAN e com rede CAN.



Fonte: http://www.cin.ufpe.br/.

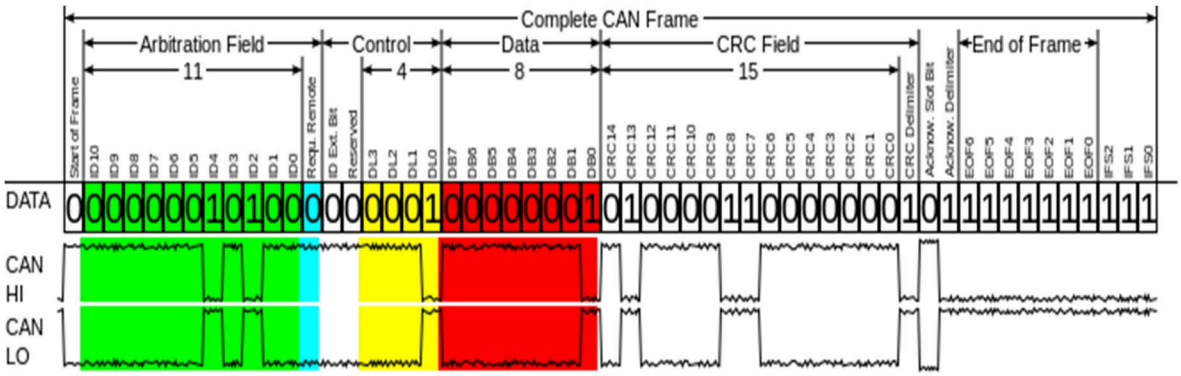
2.3.2 Formato de mensagem

Toda mensagem transmitida pelo barramento obedece a uma determinada ordem de prioridade e esta é definida pelo campo de identificação da mensagem que pode ser composta por 11 ou 29 bits, padrão e versão estendida, respectivamente. As mensagens são enviadas em frames (quadros) e cada frame contém informação sobre a origem e destinatário, sendo o *Data Frame* (frame de envio de dados – modo dominante, 0) e o *Remote Frame* (frame de pedido de informações de algum módulo – modo recessivo, 1) os mais importantes, onde o bit RTR (*Remote Transmission Request*) define qual o frame (MARQUES, 2004).

Toda mensagem possui endereçamento orientado a conteúdo, onde a variável é identificada e não o endereço da mensagem. O protocolo CAN possui atualmente duas versões: 1.0 e a 2.0 que possui divisão em 2.0A (*Standart)* ou 2.0B (estendido). Redes CAN

2.0A tem identificador de 11 bits e é possível ter até 2048 mensagens na rede enquanto a redes CAN 2.0B tem identificadores de 29 bits e assim podem ter aproximadamente 537 milhões de mensagens. Apesar de a rede CAN 2.0B resolver problemas relacionados à limitação de mensagens na rede, esta apresenta como desvantagem o maior tempo necessário para a transmissão de cada mensagem, afinal são 18 bits adicionais para identificação e isto torna-se um problema em aplicações que trabalhem em tempo real (problema conhecido como *overhead*).

Figura 13 – *Frame* do barramento CAN.



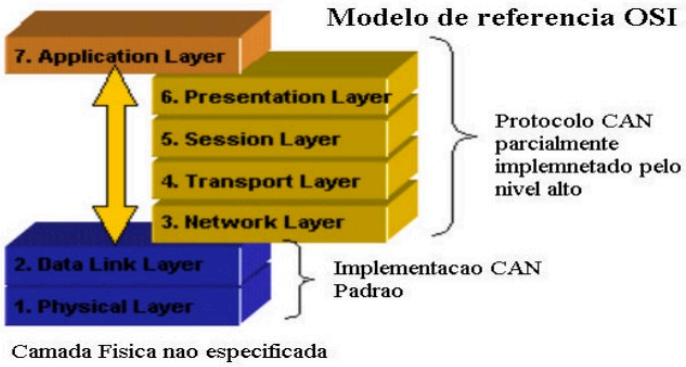
Fonte: http://machineworkshop.com.br.

2.3.3 Padrões e Normas

Duas normas se destacam na fundamentação da rede CAN, são elas: ISO11898 (características de uma rede operando com alta velocidade, 125 Kbps à 1 Mbps) e a ISO11519-2 (características de uma rede operando em baixa velocidade, 10 Kbps à 125 Kpbs). Considerando o modelo OSI de sete camadas (ISO7498), os padrões fundamentais da rede CAN estão especificados apenas para as camadas 1 e 2, física e de enlace, respectivamente, sendo as demais camadas especificadas pelos padrões abaixo (MARQUES, 2004).

- NMEA 2000: CAN 2.0B, aplicações navais e aéreas;
- SAE J1939: CAN 2.0B, aplicações automotivas;
- DIN 22783 – LBS: CAN 2.0A, aplicações agrícolas;
- ISO 11783: CAN 2.0B, também para aplicações agrícolas.

Figura 14 – Modelo OSI rede CAN.



Fonte: MARQUES, 2004.

2.3.4 Detecção de Erros

Os erros (ou falhas) são classificados em três níveis, sendo eles:

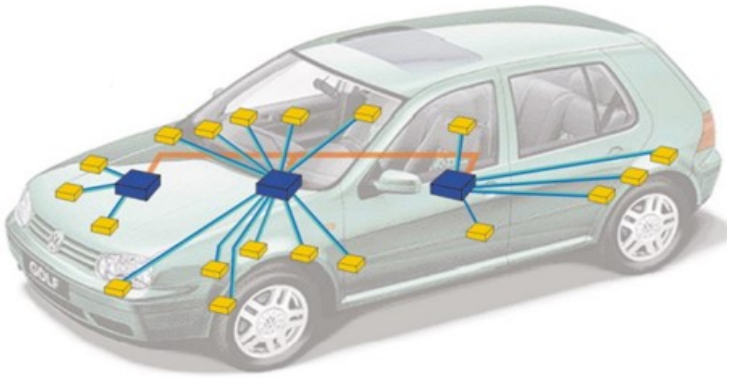
- Nível de *bit*: *Bit Monitoring*, que monitora se o barramento identifica corretamente quando há um bit dominante; Bit *Stuffing*, que verifica a quantidade de bits consecutivos e iguais, que não podem ultrapassar a quantidade de cinco unidades e caso seja necessário enviar mais de 6 bits consecutivos e idênticos, o transmissor se encarrega de acrescentar um bit de valor contrário a cada 5 bits.
- Nível de mensagem: CRC ou *Cyclic Redundancy Check*, que calcula um valor de acordo com o número de bits da mensagem e envia este valor, então o módulo receptor recalcula a quantidade de bits da mensagem e confere se é igual ao CRC; *Frame Check*, que analisa o conteúdo de alguns bits da mensagem e que não mudam de mensagem para mensagem, de acordo com o padrão CAN; ACK ou *Acknowledgment Error Check*, onde cada módulo receptor escreve um bit dominante em uma mensagem de resposta indicando que a mensagem anterior foi recebida.
- Nível físico: Caso ocorra algum problema com a fiação de dados CAN_H ou CAN_L, a rede funcionará em modo de segurança. Possíveis condições de falha são numeradas abaixo:

  - ✓ Curto do CAN_H (ou CAN_L) para o GND ou (VCC);
  - ✓ Curto entre os fios de dados CAN_H e CAN_L;
  - ✓ Ruptura do CAN_H (ou CAN_L).

2.3.5 Aspectos construtivos da rede

Qualquer rede CAN deverá apresentar terminadores da rede, onde são conectados resistores de 120 ou 124 ohm e cuja função é garantir a propagação dos sinais elétricos pelo barramento ao provocar reflexão dos sinais.

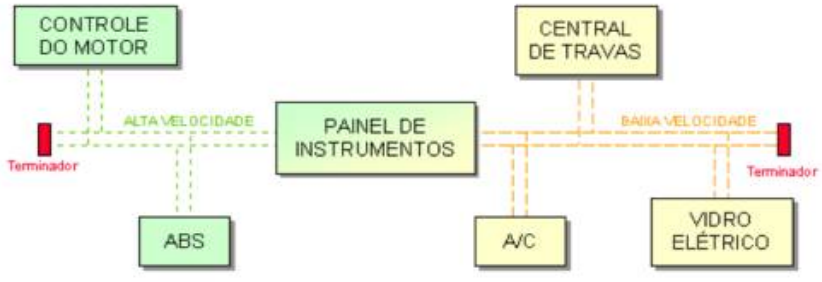
Figura 15 – Rede CAN composta por 3 ECU´s e vários módulos de controle.



Fonte: Data Exchange On The CAN Bus I.

Pode-se numa mesma rede haver duas ou mais sub-redes e estas apresentarem características diferentes, sendo a velocidade de transmissão uma das principais. Esta possibilidade torna-se útil quando se considera que em um mesmo sistema pode haver subsistemas que necessitam de maior velocidade de transmissão por envolver riscos à segurança de pessoas, a exemplo, o sistema de freios ABS. A transferência de informações entre sub-redes é realizada com a utilização de módulos (*gateways*) que operam nas duas redes. A imagem seguinte exemplifica a utilização de sub-redes, resistores de terminação, e *gateway* embarcado no painel de instrumentos do veículo automotor (PROTOCOLO CAN, 2016).

Figura 16 – Rede CAN composta por sub-redes.

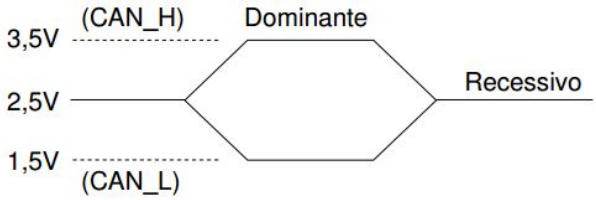


Fonte: http://machineworkshop.com.br.

2.3.6 Níveis de Tensão

O estado lógico do barramento é determinado pelo nível de tensão entre as linhas CAN_H e CAN_L, sendo considerado ***bit* recessivo** quando a diferença de tensão entre as linhas é de zero *volts* e considerado ***bit* dominante** quando há uma diferença de 2 *volts* entre os terminais da linhas. A figura abaixo mostra um exemplo (PROTOCOLO CAN, 2016).

Figura 17 – Níveis de tensão barramento CAN.

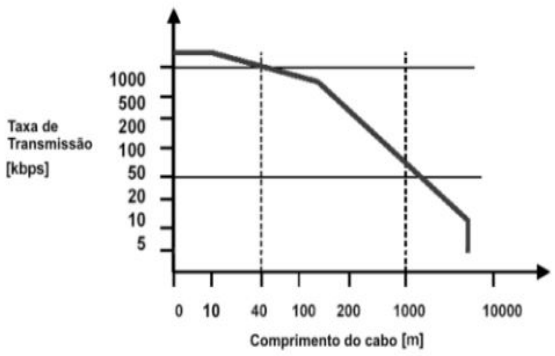


Fonte: GERALDO, 2008.

2.3.7 Velocidade x Comprimento Barramento

A velocidade de um sinal pelo barramento tem como limitante o comprimento deste. O calculo considera o tempo de transmissão de um *bit* partindo da origem e retornando a este. A figura 18 ilustra os valores máximos de comprimento por taxa de transmissão (GERALDO, 2008).

Figura 18 – Comprimento por taxa de transmissão.



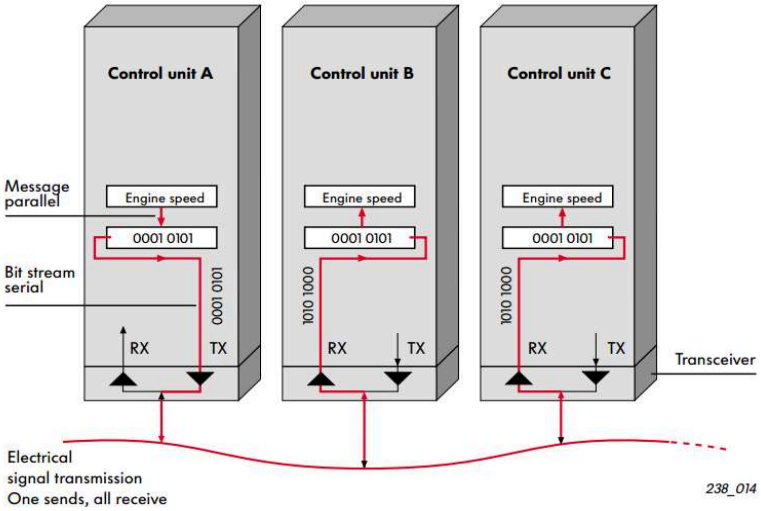
Fonte: GERALDO, 2008.

2.3.8 Controlador CAN

Os sistemas de sensores e atuadores são conectados à unidade central através do barramento CAN, porém a ponte entre eles se dá por módulos de controle, onde cada módulo

é equipado com transceptores (transmissor e receptor) CAN que transformam os dados do controlador CAN em sinais elétricos e vice-versa, transmitindo estas informações através do barramento CAN. Todos os módulos de controle recebem a mesma informação, porém só processam as que se destinam a eles. A figura abaixo mostra o processo de conversão e leitura de dados (MARQUES, 2004).

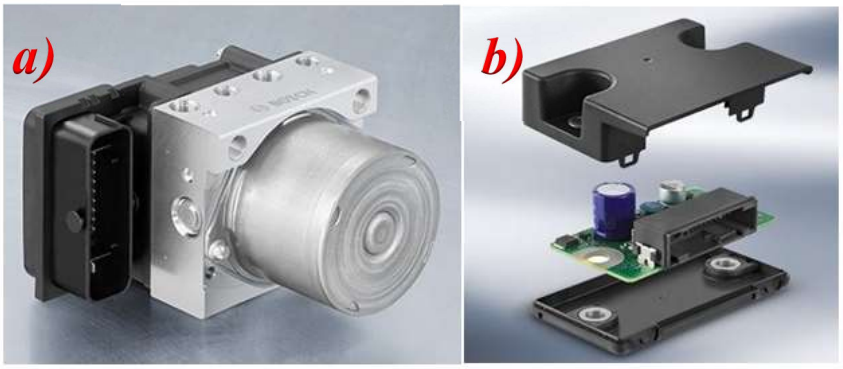
Figura 19 – Exemplificação do processo de leitura dos módulos de controle.



Fonte: Data Exchange On The CAN Bus I.

Entre os módulos de controle mais populares e presentes nos veículos estão os do sistema de ABS e *Airbag* (equipamentos suplementares de segurança passiva), que inclusive tornaram-se obrigatórios em todos os veículos novos de fabricação nacional e importados. A obrigatoriedade se deu por meio da resolução do CONTRAN Nº 311 de 03 de Abril de 2009, com prazos limites para adequação por parte de todas as montadoras.

Figura 20 – Módulos de controle dos sistemas de ABS$_{(a)}$ e *Airbag*$_{(b)}$.



Fonte: www.bosch.com.

2.3.9 ELM 327

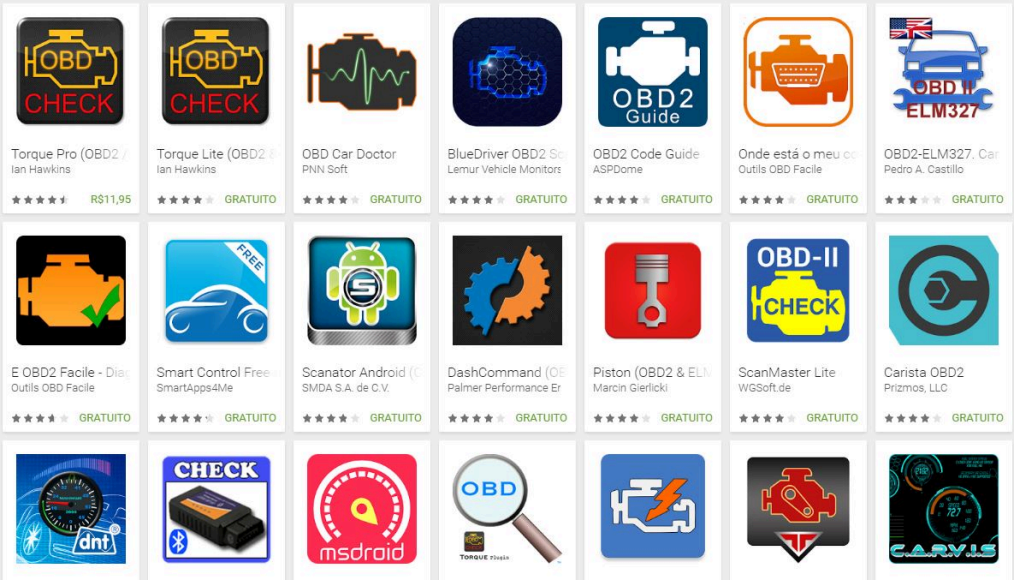
Seguindo a linha de ferramentas *scanners* para diagnósticos, tornou-se comum a comercialização de dispositivos de baixo custo capazes de requisitar parâmetros do veículo, destacando-se o ELM327. Este, diferentemente das ferramentas de scanners comumente utilizadas em oficinas, não apresenta *display* para visualização dos parâmetros, fazendo-se assim necessário o uso de microcomputadores (modelo USB). São comercializados também modelos equipados com comunicação sem fio (*wi*-fi e *bluetooth*) com foco principal na utilização por dispositivos móveis como *smartphone* e *tablets* (ELM 327, 2016).

Figura 21 – Controlador CAN ELM327, versão comunicação USB.

Fonte: https://en.wikipedia.org/wiki/ELM327.

A possibilidade de uso de *scanners* através de dispositivos móveis impulsionou o desenvolvimento de aplicativos (*apps*) para este fim, disponíveis em versões gratuitas ou não, para sistemas *Android* e iOS. No sitio https://play.google.com é possível encontrar estes aplicativos para instalação em sistemas *Android*. A figura 22 mostra exemplos de aplicativos.

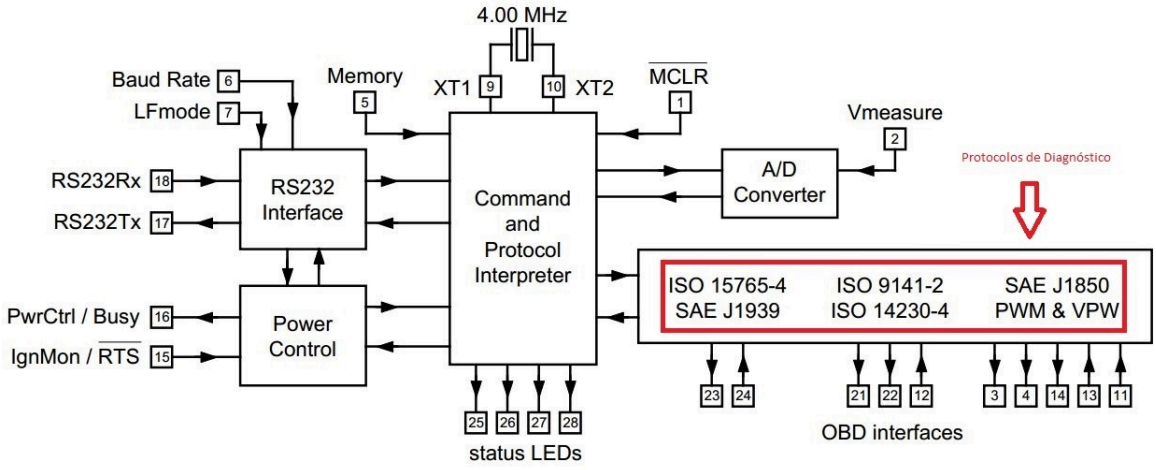
Figura 22 – Aplicativos para diagnóstico OBDII.



Fonte: https://play.google.com.

O circuito ELM327 é composto principalmente por um microcontrolador e uma interface de comunicação. O *firmware* carregado no microcontrolador processa os sinais de entrada e gera sinais de saída em conformidade com o protocolo de modulação disponível no veículo. A figura 23 exibe o diagrama de blocos do circuito com destaque para os protocolos de diagnóstico suportados.

Figura 23 – Diagrama de blocos do controlador CAN ELM327.

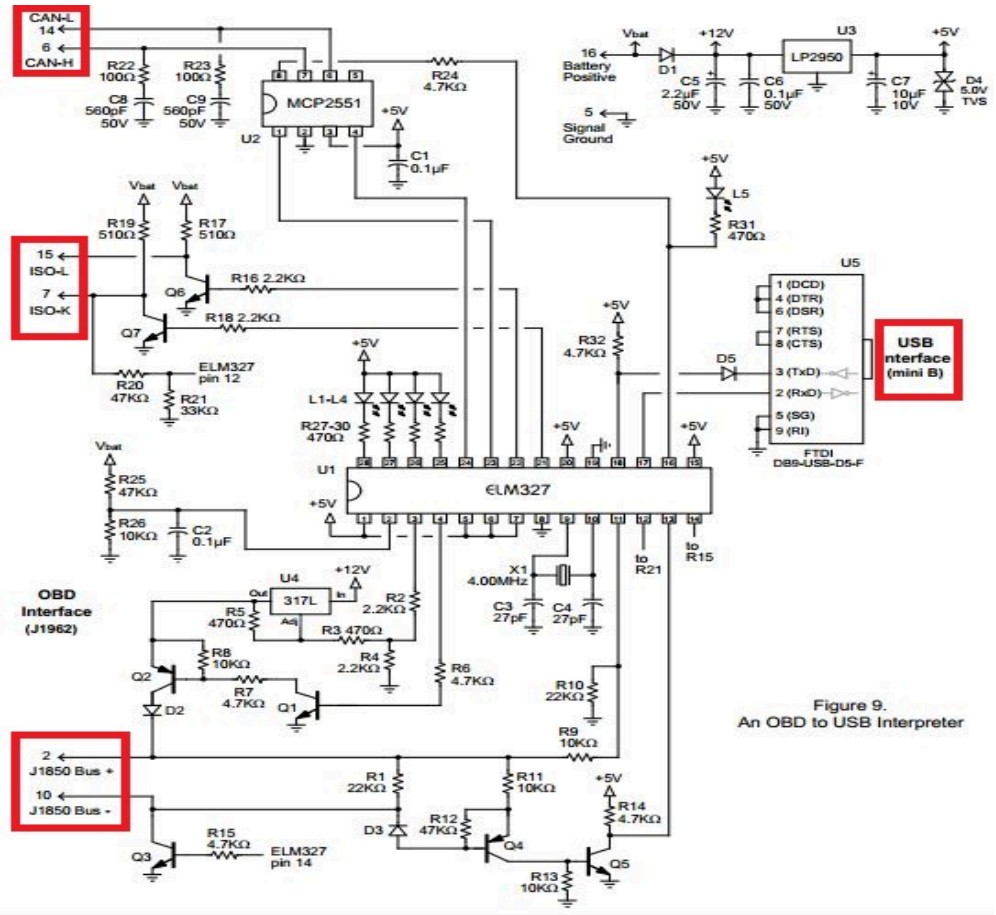


Fonte: *Datasheet* ELM327.

Basicamente o circuito microcontrolado aguarda a inserção de um parâmetro de requisição à UCE (PID) através da interface de comunicação (USB, *Wi-Fi* ou *Bluetooth*), realiza a conversão de sinais conforme o protocolo disponível e encaminha à UCE central

através do conector de diagnóstico e barramento CAN. Posteriormente, a UCE central responde à requisição seguindo caminho inverso (ELM 327, 2016).

Figura 24 – Circuito exemplo do controlador CAN ELM327.



Fonte: *Datasheet* ELM327.

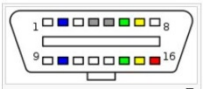
2.4 *On board diagnostics*

Desde 1994 uma rede multiplexada foi desenvolvida com o objetivo de diagnóstico da rede CAN e chamada de OBD-II, definida pela SAE J1962 e possuindo diversas opções de conexão para protocolos ISO e SAE, incluindo assim veículos de países e fabricantes diferentes. As especificações são apenas para diagnóstico automotivo. Com o desenvolvimento desta rede foram projetados equipamentos *scanners* capazes de requisitar dados de sistemas sensores e atuadores presentes nos veículos.

Através de um conector de diagnóstico é possível conectar o *scanner* e verificar dados como: pressão, temperatura, velocidade do veículo, rotação do motor, taxa de entrada de ar, nível de combustível, além de códigos de erros registrados na rede após falhas ocorridas. A figura 25 demonstra a distribuição dos pinos do conector de diagnóstico OBDII. Observa-se

que os protocolos utilizam o mesmo modelo de conector, em pinos específicos para cada protocolo. O veículo de testes utilizado no presente trabalho utiliza o protocolo de diagnóstico ISO 14230-4 KPW *fast*.

Figura 25 – Pinos conector OBDII.



1	Manufacturer discretion: • GM: J2411 GMLAN/SWC/Single-Wire CAN[20] • VW/Audi/BMW: Switched +12V to tell a scan tool whether the ignition is on. • Ford, FIAT: Infotainment CAN High[20]	9	Manufacturer discretion: • BMW: TD (Tachometer Display) signal aka engine RPM signal. • GM: 8192 bit/s ALDL where fitted. • Ford: Infotainment CAN-Low
2	Bus Positive Line of SAE J1850 PWM and VPW	10	Bus Negative Line of SAE J1850 PWM only (not SAE J1850 VPW)
3	Manufacturer discretion: • Ford: DCL(+) Argentina, Brazil (pre OBD-II) 1997-2000, USA, Europe, etc. • Ford: Medium Speed CAN-High[20] • Chrysler: CCD Bus(+)[20]	11	Manufacturer Discretion: • Ford: DCL(-) Argentina, Brazil (pre OBD-II) 1997-2000, USA, Europe, etc. • Ford: Medium Speed CAN-Low[20] Chrysler: CCD Bus(-)[20]
4	Chassis ground	12	Manufacturer discretion:
5	Signal ground	13	Manufacturer discretion: • Ford: FEPS – Programming PCM voltage
6	CAN-High (ISO 15765-4 and SAE J2284)	14	CAN-Low (ISO 15765-4 and SAE J2284)
7	K-Line of ISO 9141-2 and ISO 14230-4	15	L-Line of ISO 9141-2 and ISO 14230-4
8	Manufacturer discretion: • BMW: Second K-Line for non OBD-II (Body/Chassis/Infotainment) systems. • FIAT: Infotainment CAN-Low.	16	Battery voltage: • Type "A" 12V/4A • Type "B" 24V/2A

Fonte: https://en.wikipedia.org/wiki/On-board_diagnostics#OBD-II.

2.4.1 Protocolos

A rede de bordo dedicada ao diagnóstico (OBDII) veicular utiliza variados protocolos, específicos para diagnóstico e diferenciam-se por fabricante e/ou região do globo. A figura 26 exibe os mais utilizados.

Figura 26 – Protocolos de diagnósticos.

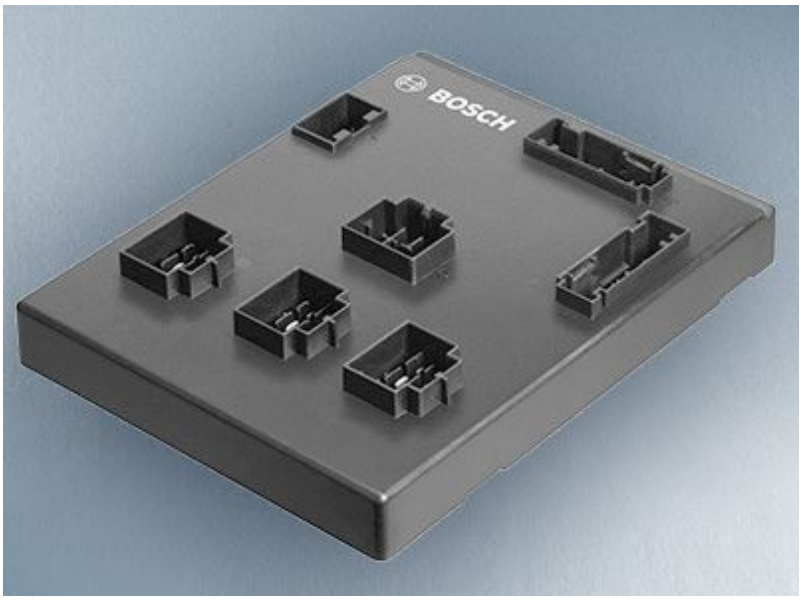
NOME:	USUÁRIO:	USO:	MODELOS ANOS:	COMENTÁRIOS:
ISO 15765-4	EUROPA	E-OBD	2000+	E-ODB CAN
ISO 15765-4	TODOS	ODB-III	2007+	OBD HARMONIZADO
J 1850	GM, FORD, DC	ODB-II	1994+	NÃO ACEITO NA EUROPA
ISO 9141-2	EUROP, ASIA, ALGUMAS NOS E.U.A.	ODB-II	1994+	VELHO OBD-II UART
ISO 14230-4	MUITAS	OBDII, ODB-III	2000+	KEYWORD 2000

Fonte: MARQUES, 2004.

2.4.2 Parâmetros de Diagnósticos

As mensagens de diagnóstico são transmitidas utilizando tecnologias de modulação pulsada de acordo com o protocolo presente no veículo. A leitura de parâmetros de diagnósticos poderá então ser realizada com a utilização de ferramentas de diagnósticos conhecidas como *scanners*, onde o operador insere parâmetros e aguarda a resposta. Cada veículo disponibiliza uma diversidade de informações relacionadas aos sensores e atuadores presentes, mas a variedade de informações disponíveis difere entre montadoras.

Figura 27 – Módulo de controle de sensores e atuadores fabricação Bosch.



Fonte: www.bosch.com.

Os parâmetros são divididos em modos que se estendem de 01_{16} a $0A_{16}$ (OBD-II *Standard* SAE J1979), mas nem todos os veículos possuem suporte a todos os modos. A solicitação de informações é realizada inserindo o modo e o parâmetro relativo ao sistema a ser consultado, em formato hexadecimal. A SAE *Standard* J1939 define então a faixa de parâmetros (PID´s) genéricos a serem seguidos pelas montadoras e inclui as faixas de PID´s reservados para uso específicos pelas montadoras. A tabela 3 mostra as funções de cada modo de operação.

Tabela 3 – Modos de operação para diagnóstico.

Modo (hexadecimal)	**Descrição**
01	Dado atual
02	Dado armazenado
03	Códigos de diagnóstico armazenados
04	Zerar códigos de falha
05	Resultados de testes, monitoramento de sensor de oxigênio
06	Resultados de testes de outros componentes monitorados pelo sistema
07	Mostrar códigos de diagnósticos pendentes
08	Operação de controle dos componentes à bordo
09	Solicitação de informações do veículo
0A	Códigos de falhas de problemas permanentes

Fonte: https://en.wikipedia.org/wiki/OBD-II_PIDs.

O modo 01 apresenta o maior número de parâmetros para consulta, mas nem todos estão disponíveis em todos os veículos. A tabela 4 exibe exemplos de alguns dos principais parâmetros do modo 01 e que estão presentes na maioria dos veículos nacionais e importados.

Tabela 4 – Parâmetros de diagnóstico em modo 01.

PID (Hex)	**Bytes de Retorno**	**Descrição**	**Mín.**	**Máx.**	**Unid.**	**Fórmula**
04	1	Carga sobre o motor	0	100	%	A/2.55
05	1	Temperatura arrefecimento	-40	215	ºC	A-40
0C	2	Rotação do motor	0	16.383,75	rpm	[(256*A)+B]/4
0D	1	Velocidade do veículo	0	255	km/h	A
33	1	Pressão barométrica absoluta	0	255	kPa	A
5C	1	Temperatura óleo do motor	-40	210	ºC	A-40
5E	2	Taxa de consumo de combustível	0	3212.75	L/h	[(256*A)+B]/20
42	2	Tensão do módulo (ECU principal)	0	65.535	V	[(256*A)+B]/1000

“>”, indica sistema pronto para receber parâmetro; “A”, terceiro *byte* de resposta; “B”, quarto *byte* de resposta.

Ex.: Solicitar o parâmetro rotação do motor.

>01 0C //Solicitação, onde “01” é o modo, e 0C o parâmetro de rotação do motor.
>41 0C 1D 0C //Resposta à solicitação, onde os bytes “41 0C” confirmam qual parâmetro foi solicitado e os bytes 1D 0C são a resposta à solicitação, respectivamente representam “A” e “B”.

Fonte: https://en.wikipedia.org/wiki/OBD-II_PIDs.

3 METODOLOGIA

3.1 Materiais e métodos

De acordo com Gil (1999 *apud* ALMEIDA, 2008, p.7), a pesquisa tem um caráter pragmático, é um "processo formal e sistemático de desenvolvimento do método científico. O objetivo fundamental da pesquisa é descobrir respostas para problemas mediante o emprego de procedimentos científicos".

Ainda neste contexto, Lakatos (2001) afirma que o método de procedimento é uma etapa mais concreta da investigação, uma vez que pressupõe uma atitude real em relação ao fenômeno.

Assim, a pesquisa obedecerá aos seguintes aspectos metodológicos:

- Quanto à natureza, será uma pesquisa aplicada, pois será dirigida à solução de problemas específicos;
- Quanto à forma de abordagem do problema, será uma pesquisa quantitativa, pois permitirá que as informações sejam quantificadas para melhor análise dos resultados;
- Quanto aos objetivos, será uma pesquisa explicativa, pois fará uso de métodos experimentais;
- Quanto ao procedimento técnico, será bibliográfica e experimental, pois haverá consulta à literatura já publicada, bem como definição de objeto de estudo e variáveis a serem controladas.

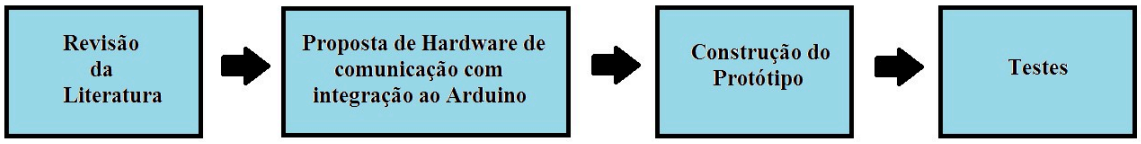
Portanto, a pesquisa objeto deste trabalho foi desenvolver uma unidade de comando eletrônico para integração à rede CAN automotiva e inserir a UFERSA nesta área de pesquisa global.

3.2 Procedimentos da pesquisa

Inicialmente, na primeira etapa da pesquisa, realizou-se revisão da literatura acerca das unidades de comando eletrônico presentes nos veículos automotores, suas arquiteturas e características de funcionamento, distribuição no veículo e histórico de evolução. Em seguida, foi feita revisão bibliográfica cerca da rede de bordo CAN e dos principais protocolos de comunicação utilizados nesta, assim como suas características de implementação e operação.

Numa segunda etapa, foram levantados dados relacionados aos principais sistemas embarcados nos veículos e gerenciados pela rede de bordo. Em seguida, houve a identificação dos procedimentos necessários para estabelecer comunicação com a UCE central e requisitar informações sobre os sistemas embarcados. Com estas informações, foi proposta e desenvolvida uma estrutura de *hardware* com e implementação de *firmware* para gerenciamento da comunicação com a UCE central do veículo de testes, onde solicitações pudessem ser enviadas à UCE principal e a mesma retornasse informações relativas aos sensores e atuadores do veículo em questão. Os dados, então seriam analisados pelo firmware desenvolvido e amostrados em visor de cristal líquido para, em testes de campo, realizar a validação dos resultados através de comparação com mostradores disponíveis no próprio painel do veículo de testes ou utilizando instrumentos de medição específicos.

Figura 28 – Fluxograma das etapas de elaboração desta pesquisa.



Fonte: Próprio Autor.

4 ESPECIFICAÇÃO DA UCE

Este capítulo descreve os materiais e métodos a serem utilizados para o desenvolvimento da UCE, permitindo obter informações sobre cada componente necessário ao desenvolvimento do protótipo. A escolha dos materiais para uso no desenvolvimento do projeto se dá na forma de sugestão para os que vierem a executá-los, sendo de escolha do usuário a utilização ou não dos componentes, bem como de maiores adequações, baseadas na especificidade dos equipamentos adquiridos ou disponíveis para tal. Observa-se ainda que os critérios de escolha dos componentes foram baseados no custo-benefício, disponibilidade, e capacidade para execução da ação pretendida, eliminando assim a necessidade de maiores complexidades de cálculos, mas que virão a ser levantados como proposta para trabalhos futuros.

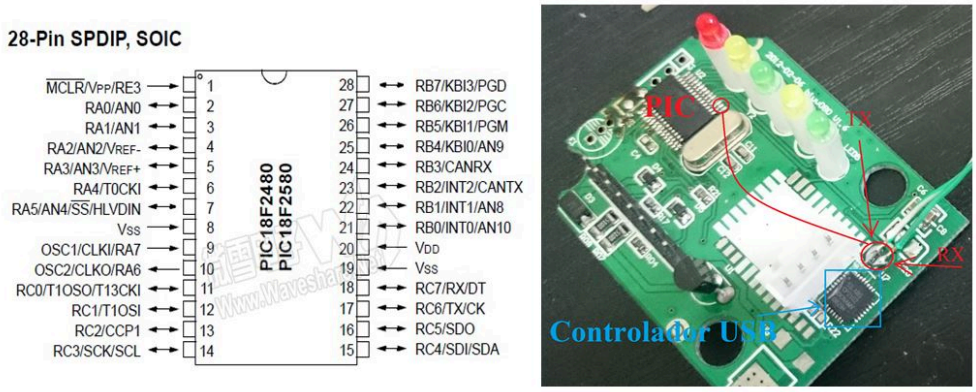
4.1 Aquisição de dados

4.1.1 ELM 327

Como forma de estabelecer a comunicação entre o protótipo e a UCE central do veículo foi selecionado o controlador CAN ELM327 ao considerar, principalmente, a flexibilidade de uso em diversos modelos e fabricantes de veículos, uma vez que o mesmo suporta diversos protocolos de comunicação de diagnóstico além de baixo custo de aquisição em relação aos modelos de *scanners* comerciais disponíveis. Por apresentar comunicação externa via conexão USB e a forma escolhida de comunicação com o Arduino ser serial UART, o modelo ELM327 precisou de adequações.

O microcontrolador presente no controlador CAN dispõe em seus pinos 17(TX) e 18(RX) de opção para conexão serial UART e então para que pudessem ser conectados ao Arduino foram soldados fios na saída do microcontrolador, antes de passar pelo circuito de controle USB-Serial. Assim, todo tráfego de dados se dará entre o Arduino e o microcontrolador PIC18F2580, isolando a comunicação USB.

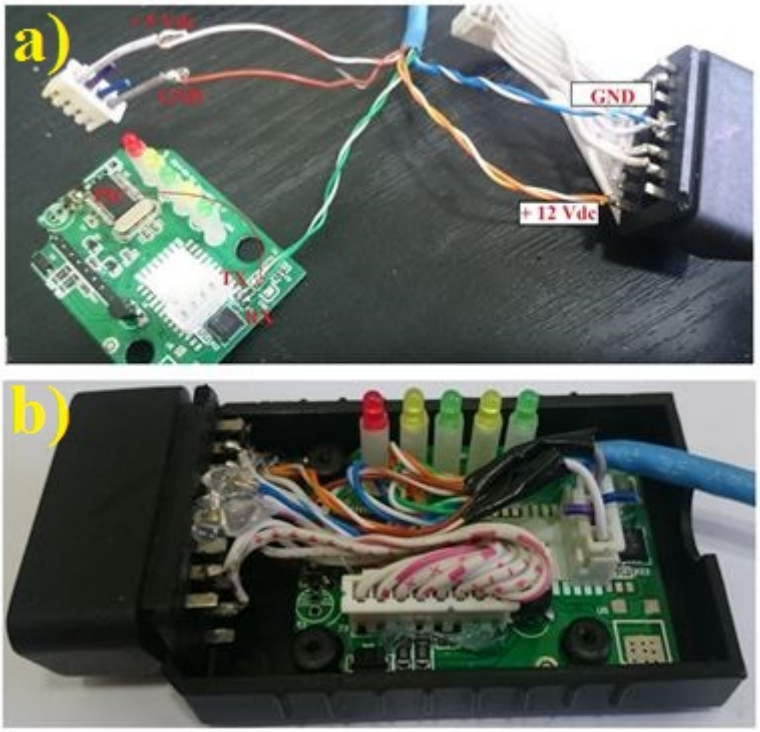
Figura 29 – Conexão para comunicação serial UART.



Fonte: Esq. (www.waveshare.net), direita (autor).

Para possibilitar a conexão entre o controlador CAN e o Arduino foi utilizado um cabo par trançado padrão CAT5e 8 vias com a função de alimentação e transmissão de dados. A alimentação 12Vdc e GND é fornecida pelo veículo por meio do conector de diagnóstico, em seus pinos 16 e 4, e então direcionada ao Arduino que por sua vez regula a tensão em +5Vdc e a envia até a placa do controlador CAN, energizando-a sob mesmo referencial.

Figura 30 – a) Conexões físicas; b) Montagem do cabo de alimentação/dados.



Fonte: Próprio autor.

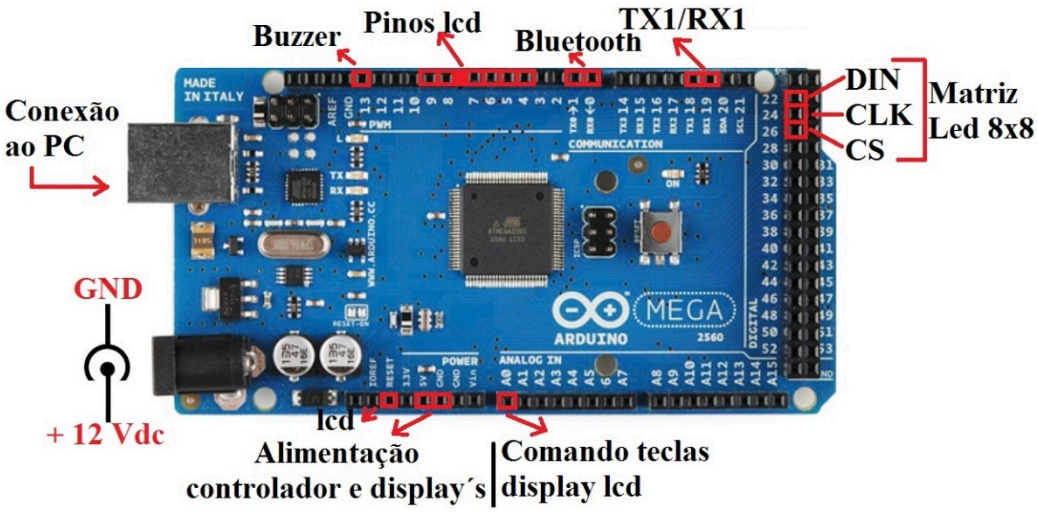
4.2 Processamento

O controlador CAN funciona apenas como um canal de comunicação entre o computador do veículo e o ambiente externo, logo qualquer ação de processamento e lógica para funcionamento de componentes externos como sensores e atuadores necessitará de algum sistema de controle. A plataforma Arduino, como já mencionada anteriormente, exercerá a função de direcionar ao controlador CAN os parâmetros de requisição de informação sobre os sistemas embarcados no veículo, processará a resposta e executará uma ação pré-definida. O modelo de placa escolhida entre as disponíveis na plataforma Arduino foi o MEGA 2560 e a escolha se deu principalmente em virtude da:

- Quantidade de portas de entrada e saída tanto analógicas quanto digitais;
- Quantidade de portas de comunicação serial;
- Excelente relação custo-benefício considerando a capacidade de processamento;
- Quantidade de memória não-volátil disponível.

Os pinos utilizados do Arduino MEGA para este projeto obedecem à figura 31 e devido a grande quantidade de portas disponíveis, neste mesmo módulo poderão ser conectados diversos dispositivos sensores e atuadores para comando em função dos dados obtidos da UCE central.

Figura 31 - Pinos utilizados do Arduino MEGA 2560.



Fonte: Próprio autor.

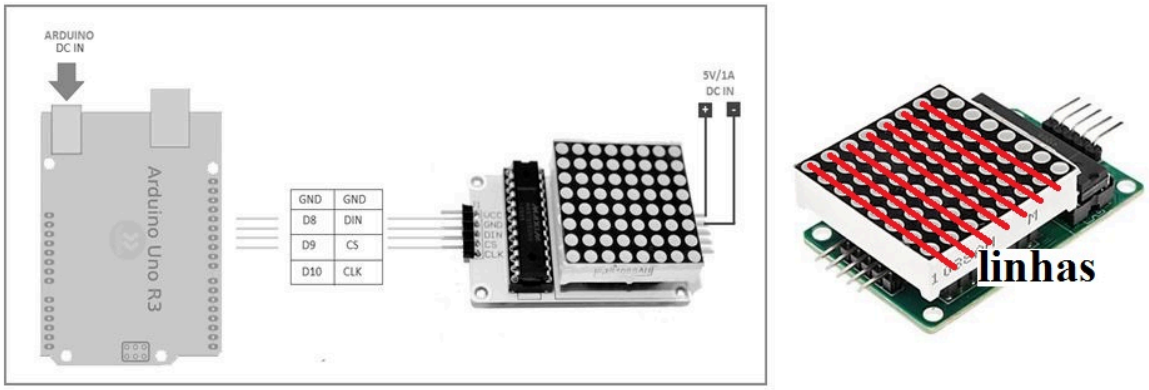
4.3 Saída

4.3.1 Módulo Matriz *Led* 8x8

Respostas às requisições são direcionadas ao Arduino por meio de uma das portas de comunicação serial (TX1/RX1), então processadas e a informação disponibilizada através de um *display* lcd 16x2. Um segundo *display*, este composto por uma matriz de *led´s* 8x8 e controlado pelo protocolo de comunicação SPI (*Serial Peripheral Interface*), é então incorporado para permitir melhor visualização das variáveis em termos de máximos e mínimos instantâneos.

O módulo matriz de *led´s* utiliza protocolo SPI por meio do circuito integrado MAX7219 e assim reduz o número de portas do Arduino utilizadas durante a operação. A lógica implementada para este módulo foi definida para utilização em qualquer dos parâmetros selecionados para monitoramento e acenderá em níveis. São oito níveis representados por oito *led´s* cada. Tomando por exemplo o parâmetro rotação do motor, com um *range* que abrange valores de 0 a 7000 RPM (painel veículo de testes), estes são então divididos entre os níveis disponíveis no módulo.

Figura 32 – Conexão módulo display led´s a uma placa Arduino UNO.



Fonte: http://www.electroschematics.com/10510/arduino-8-8-led-matrix/.

Um segundo exemplo de aplicação para o módulo matriz de *led´s* se dá no monitoramento do parâmetro de carga sobre o motor. Este parâmetro não está disponível para visualização no painel do veículo de testes, mas mostra-se um importante dado a ser verificado para acompanhamento do desempenho do motor em cada instante do trajeto. Este parâmetro está ainda diretamente relacionado ao modo como se dirige o veículo, onde o melhor desempenho

dependerá de fatores como aceleração adequada e engate da marcha correta para a rotação que se encontra o motor.

4.3.2 Módulo Matriz *Led* 8x8

O módulo (*shield*) display de cristal líquido (lcd) adicionado suporta a inserção de até 32 caracteres simultâneos e distribuídos em 2 linhas e 16 colunas, ou 16x2 como é comercialmente descrito. No projeto, foi inserido para as seguintes funções:

- Permitir ao usuário que este selecione o parâmetro que deseja monitorar e para tal faz uso das teclas disponíveis, sendo: *LEFT*, retornar menu; *RIGHT*, avançar menu de opções; *RESET*, para reiniciar o *firmware*;
- Visualizar através do *display* o parâmetro selecionado, bem como os valores obtidos a cada instante.

Figura 33 – *Display* lcd, rotina de inicialização e menu do protótipo.

Fonte: *Shield* lcd (www.flipflop.com); operação do *display*, próprio autor.

4.3.3 *Buzzer*

Com o intuito de demonstrar mais um exemplo de aplicação que utilize os dados fornecidos pela UCE central foi inserido um *buzzer* (emissor de sinal sonoro) e configurado para que alerte o motorista quando houver ultrapassado a velocidade máxima de 80 km/h.

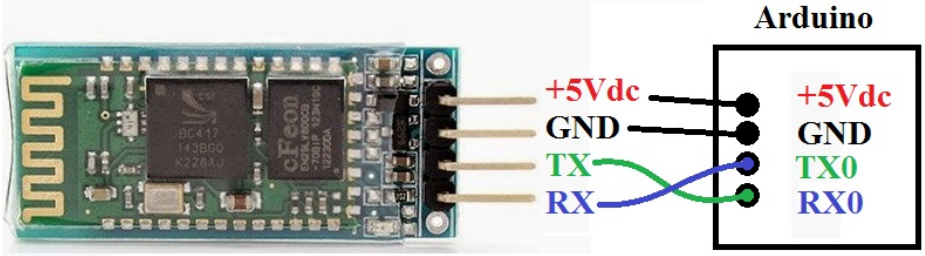
Figura 34 – Módulo *buzzer* e descrição da conexão dos pinos.

Fonte: www.mercadolivre.com.

4.3.4 *Bluetooth*

Expandindo as funcionalidades do protótipo de UCE, foi inserido ainda o módulo de comunicação *bluetooth* modelo HC-06. Este por sua vez, depois de pareado com outro dispositivo, enviará informações sobre os parâmetros selecionados no teclado do módulo lcd.

Figura 35 – Módulo *bluetooth*.



Fonte: www.arduinoecia.com.br.

Para testes de funcionamento foi utilizado aplicativo com interface de comunicação serial. Um dispositivo móvel, com sistema *Android* instalado e *hardware Bluetooth* nativo, estabelece a conexão com o módulo *Bluetooth* presente na UCE (TX0/RX0 do Arduino) e então o aplicativo gerencia o tráfego de dados serial.

Figura 36 – Aplicativo *Android bluetooth* terminal.

Fonte: https://play.google.com/store.

5 RESULTADOS E DISCUSSÕES

Apesar da grande quantidade de modelos de veículos e montadoras, reunir informações mais detalhadas sobre as UCE´s ainda é um grande desafio. Muito se deve ao ambiente competitivo em que se encontram, obrigando-as a buscar a confidencialidade de suas arquiteturas e sistemas. Muito se tem avançado e a arquitetura fechada e repleta de obscuridade tem sido liberada como contrapartida à possibilidade de inovação na implementação de novos serviços ofertados em cada veículo.

A figura 37 mostra o protótipo desenvolvido mediante utilização dos componentes apresentados ao longo deste trabalho. Pode-se observar a utilização de uma caixa transparente para acomodação dos componentes, posição escolhida para inserção do display lcd, módulo matriz de *led´s*, assim como também o espaço reservado para instalação do *módulo bluetooth* e *buzzer*.

Figura 37 – Protótipo de UCE.

Fonte: Próprio autor.

O veículo utilizado na realização dos testes é detalhado a seguir:

- Caminhonete da marca Toyota;
- Modelo *Hilux*, cabine dupla, 4x2 SR;
- Potência 158 CV, combustível gasolina/GNV, ano 2009/2009;
- Protocolo OBD: ISO 14230-4 KWP *fast*.

Como forma de validar o protótipo, alguns parâmetros de diagnóstico foram selecionados para monitoramento e então implementados em *firmware* na UCE. Apenas o modo de operação 01 foi testado devido ao grande número de variáveis disponíveis, porém há a possi-

bilidade de implementar outros modos de operação, principalmente o modo 08, pois este opera os sistemas a bordo do veículo.

Dentre os parâmetros presentes no modo de operação 01, há genéricos e os que são reservados para uso pelas montadoras, assim cada uma equipa seus veículos com os mais variados tipos de sensores e/ou atuadores, selecionando PID´s específicos. Para realização dos testes, o *firmware* foi implementado considerando a utilização dos parâmetros abaixo listados.

- Menu 01: Parâmetro $(01\ 04)_{\text{base 16}}$ – Carga sobre o Motor. Retorno $(41\ 04\ XX)_{\text{base 16}}$;
- Menu 02: Parâmetro $(01\ 0C)_{\text{base 16}}$ – Rotação do Motor. Retorno $(41\ 0C\ XX\ YY)_{\text{base 16}}$;
- Menu 03: Parâmetro $(01\ 05)_{\text{base 16}}$ – Temperatura arrefecimento. Retorno $(41\ 05\ XX)_{\text{base 16}}$;
- Menu 04: Parâmetro $(01\ 11)_{\text{base 16}}$ – Posição do acelerador. Retorno $(41\ 11\ XX)_{\text{base 16}}$;
- Menu 05: Parâmetro $(01\ 0D)_{\text{base 16}}$ – Velocidade do veículo. Retorno $(41\ 0D\ XX)_{\text{base 16}}$;
- Menu 06: Parâmetro $(01\ 33)_{\text{base 16}}$ – Pressão barométrica absoluta. Retorno $(41\ 33\ XX)_{\text{base 16}}$;
- Menu 07: Parâmetro $(01\ 42)_{\text{base 16}}$ – Tensão UCE. Retorno $(41\ 42\ XX\ YY)_{\text{base 16}}$;
- Menu 08: Parâmetro $(01\ 5E)_{\text{base 16}}$ – Taxa de combustível. Retorno $(41\ 5E\ XX\ YY)_{\text{base 16}}$;

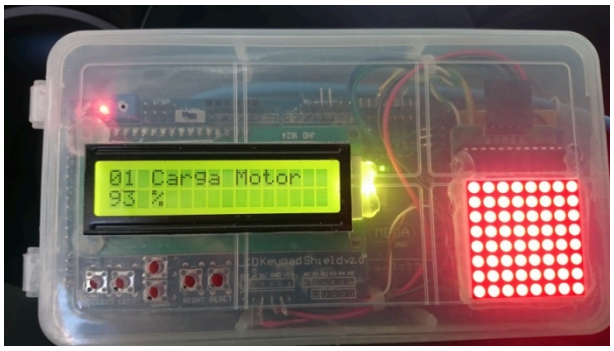
A seguir é possível observar imagens capturadas durante a realização dos testes em campo e obter maiores informações sobre o funcionamento do protótipo de UCE, objeto deste trabalho. O apêndice deste trabalho apresenta o código fonte do *firmware* implementado para funcionamento do protótipo.

5.1 Carga sobre o motor

O primeiro índice do menu refere-se ao parâmetro de cálculo da carga sobre o motor, cujo código de requisição PID é o $04_{\text{base 16}}$ e retorna como resposta um valor hexadecimal de apenas um *byte*. O valor recebido deve então ser convertido para o sistema decimal e aplicado à equação padrão ($\frac{A_{base10}}{2{,}55_{base10}}$), em que "A" representa o valor do *byte* recebido e já convertido. O valor obtido do cálculo representa então a carga instantânea sobre o motor e este pode variar de zero a 100%. Este parâmetro mostra-se de grande importância, pois reflete, entre outras questões, a forma como o usuário pilota o veículo. Trocas de câmbio manual fora do tempo correto (aceleração ou rotação indevidas para a marcha em questão) provocam sobrecarga ao

motor desnecessariamente, reduzindo a eficiência da relação torque x consumo. A figura 38 exibe uma das leituras realizadas em testes de campo, onde o parâmetro em questão apresentou carga de 93% sobre o veículo de testes, sendo ainda amostrado através do uso do módulo *display* de *led´s* que serve de monitor de carga, auxiliando o piloto quanto ao esforço (carga) aplicado ao motor.

Figura 38 – Protótipo de UCE, medição da carga sobre o motor.



Fonte: Próprio autor.

5.2 Rotação do motor

O segundo índice do menu refere-se ao parâmetro de rotação do motor, expresso em RPM (Rotações por minuto) cujo código de requisição PID é o $0C_{\text{base 16}}$ e retorna como resposta um valor hexadecimal composto por dois *bytes*. O valor recebido deve então ser convertido para o sistema decimal e aplicado à equação padrão ($\frac{256_{base10}A_{base10}+B_{base10}}{4_{base10}}$), em que "*A*" e "*B*" representam o valor dos bytes recebidos e já convertidos. O valor obtido do cálculo será instantâneo e poderá variar entre zero e $16.383{,}75_{\text{base 10}}$ RPM. A figura 39 exibe uma das leituras realizadas em testes de campo, onde o parâmetro em questão apresentou $1.996{,}0_{\text{base 10}}$ RPM. O módulo display de led´s, que é comandado pelo protótipo de UCE através do *firmware* implementado, foi configurado para amostrar resultados entre zero e $7.000{,}0_{\text{base 10}}$ RPM, pois este é o valor máximo admissível pelo painel de instrumentos do veículo de testes. A discrepância entre o valor obtido pelo protótipo de UCE e o painel de instrumentos se deu em virtude da imprecisão causada pelo fundo de escala do painel de instrumentos do veículo de testes.

Apesar do painel de instrumentos do veículo já exibir os valores de rotação do motor, este parâmetro foi implementado no protótipo de UCE devido à possibilidade de uso da variável no equacionamento de soluções a serem desenvolvidas em trabalhos futuros.

Figura 39 – Protótipo de UCE, medição da rotação do motor.

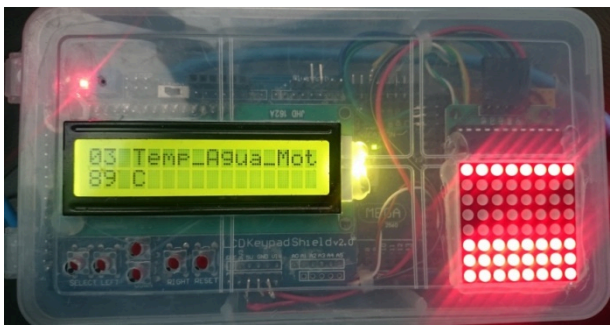
Fonte: Próprio autor.

5.3 Temperatura de arrefecimento

O terceiro índice do menu refere-se ao parâmetro de temperatura de arrefecimento do motor (informação do sensor em contato com a água do radiador e que circula pelo bloco do motor), cujo código de requisição PID é o $05_{\text{base }16}$ e retorna como resposta um valor hexadecimal de apenas um *byte*. O valor recebido deve então ser convertido para o sistema decimal e aplicado à equação padrão ($A_{base10} - 40$), em que "A" representa o valor do *byte* recebido e já convertido. O valor obtido do cálculo representa então a temperatura em graus Celsius, podendo esta variar entre $-40{,}0_{\text{base }10}$ e $215{,}0_{\text{base }10}$ ºC, valor instantâneo. A figura 40 exibe uma das leituras realizadas em testes de campo, onde o parâmetro em questão apresentou a temperatura de 89,0 ºC sendo ainda mostrado através do uso do módulo *display* de *led's*.

Obter, sempre que necessário, o valor instantâneo deste parâmetro torna-se importante, por exemplo, para a implementação de recursos (sinais sonoros, etc.) que alertem o piloto sobre possível superaquecimento do motor.

Figura 40 – Protótipo de UCE, medição de temperatura de arrefecimento.

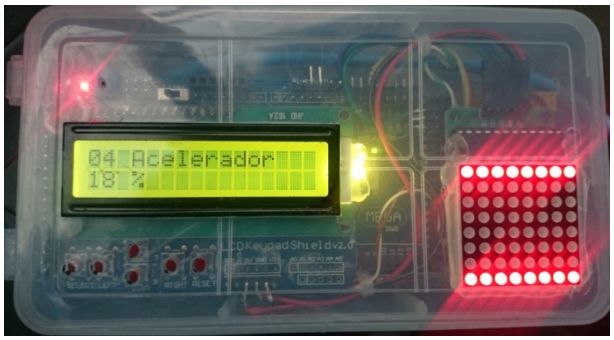


Fonte: Próprio autor.

5.4 Posição do acelerador

O quarto índice do menu refere-se ao parâmetro de posição do acelerador do motor, cujo código de requisição PID é o $11_{\text{base }16}$ e retorna como resposta um valor hexadecimal de apenas um *byte*. O valor recebido deve então ser convertido para o sistema decimal e aplicado à equação padrão ($\frac{A_{base10}}{2{,}55_{base10}}$), em que "*A*" representa o valor do *byte* recebido e já convertido. O valor obtido do cálculo representa então a posição relativa do módulo de aceleração e sua variação é percentagem, com valores instantâneos entre zero e 100%. A figura 41 exibe uma das leituras realizadas em testes de campo, onde o parâmetro em questão apresentou como resposta 18% da máxima aceleração disponível, sendo ainda amostrado através do uso do módulo *display* de *led's*. Observou-se ainda que o valor de 18% de aceleração, no veículo de testes, representou a aceleração mínima configurada. Desse modo, permanecendo o veículo em estado de ponto morto, sem intervenção no pedal acelerador, esta aceleração deve se manter constante, pois a UCE central monitora o estado dos sistemas diversos e ajusta-os conforme necessário para maior eficiência do veículo.

Figura 41 – Protótipo de UCE, medição da posição do acelerador.



Fonte: Próprio autor.

5.5 Velocidade do veículo

O quinto índice do menu refere-se ao parâmetro de velocidade do veículo, cujo código de requisição PID é o $0D_{\text{base }16}$ e retorna como resposta um valor hexadecimal de apenas um *byte*. O valor recebido deve então ser convertido para o sistema decimal e este será de modo direto, atribuído como o valor da velocidade instantânea, $A_{\text{base }10}$, em que "A" representa o valor do *byte* recebido e já convertido, podendo apresentar valores que podem variar de zero a $255{,}0_{\text{base }10}$ km/h. A figura 42 exibe uma das leituras realizadas em testes de campo, onde o

parâmetro em questão apresentou resposta de 19 km/h, sendo ainda amostrado através do uso do módulo *display* de *led's*. Observou-se ainda que o valor apresentado aparentemente divergiu do valor amostrado pelo painel de instrumentos do veículo de testes em 1,0 km/h, porém isto foi evidenciado devido à imprecisão causada pela fundo de escala do mostrador analógico do painel de instrumentos.

Como exemplo de recurso que utilize este parâmetro, foi implementado no *firmware* do protótipo de UCE um controle de velocidade do veículo, em que caso o veículo ultrapasse os 80km/h será emitido um sinal sonoro (bip) de alerta para que o piloto reduza então a velocidade.

Figura 42 – Protótipo de UCE, medição da velocidade do veículo.

Fonte: Próprio autor.

5.6 Pressão barométrica

O sexto índice do menu refere-se ao parâmetro de pressão barométrica, cujo código de requisição PID é o $33_{base\ 16}$ e retorna como resposta um valor hexadecimal de apenas um *byte*. O valor recebido deve então ser convertido para o sistema decimal e este será de modo direto, atribuído como o valor da pressão barométrica instantânea, $A_{base\ 10}$, em que "A" representa o valor do byte recebido e já convertido, podendo apresentar valores que podem variar de zero a $255{,}0_{base\ 10}$ kPa (QuiloPascal). A figura 43 exibe uma das leituras realizadas em testes de campo, onde o parâmetro em questão apresentou resposta de 97 kPa, sendo ainda amostrado através do uso do módulo *display* de *led's*.

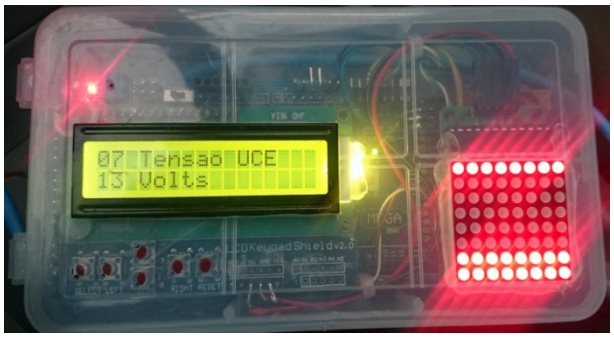
Figura 43 – Protótipo de UCE, medição da pressão barométrica.

Fonte: Próprio autor.

5.7 Tensão aplicada ao módulo de controle

O sétimo índice do menu refere-se ao parâmetro de tensão aplicada ao módulo de controle (UCE principal), expresso em volts (V) cujo código de requisição PID é o $42_{\text{base }16}$ e retorna como resposta um valor hexadecimal composto por dois *bytes*. O valor recebido deve então ser convertido para o sistema decimal e aplicado à equação padrão ($\frac{256_{base10}A_{base10}+B_{base10}}{1000_{base10}}$), em que "A" e "B" representam o valor dos *bytes* recebidos e já convertidos. O valor obtido do cálculo será instantâneo e poderá variar entre zero e $65.535{,}0_{\text{base }10}$ V. A figura 44 exibe uma das leituras realizadas em testes de campo, onde o parâmetro em questão apresentou $13{,}0_{\text{base }10}$ V como resposta à requisição, sendo ainda amostrado através do uso do módulo *display* de *led's*.

Figura 44 – Protótipo de UCE, medição da tensão de entrada na UCE principal.



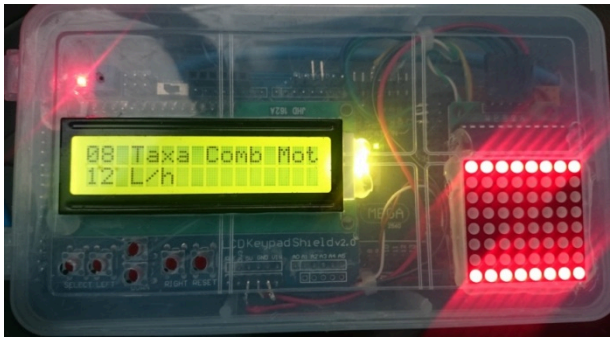
Fonte: Próprio autor.

5.8 Taxa de consumo

O oitavo e último índice implementado do menu refere-se ao parâmetro da taxa de consumo de combustível do motor, expresso em litros por hora (L/h) cujo código de requisição PID é o $5E_{base\ 16}$ e retorna como resposta um valor hexadecimal composto por dois *bytes*. O valor recebido deve então ser convertido para o sistema decimal e aplicado à equação padrão ($\frac{256_{base10}A_{base10}+B_{base10}}{20_{base10}}$), em que "A" e "B" representam o valor dos *bytes* recebidos e já convertidos. O valor obtido do cálculo será instantâneo e poderá variar entre zero e $3.212{,}75_{base\ 10}$ L/h. A figura 45 exibe uma das leituras realizadas em testes de campo, onde o parâmetro em questão apresentou consumo de $12{,}0_{base\ 10}$ L/h como resposta à requisição, sendo ainda amostrado através do uso do módulo *display* de *led's*.

Como sugestão para implementação em trabalhos futuros, pode-se gerar gráficos com amostragem das taxas instantâneas de consumo, bem como cálculos de consumo médio do veículo.

Figura 45 – Protótipo de UCE, medição da taxa de consumo de combustível pelo motor.



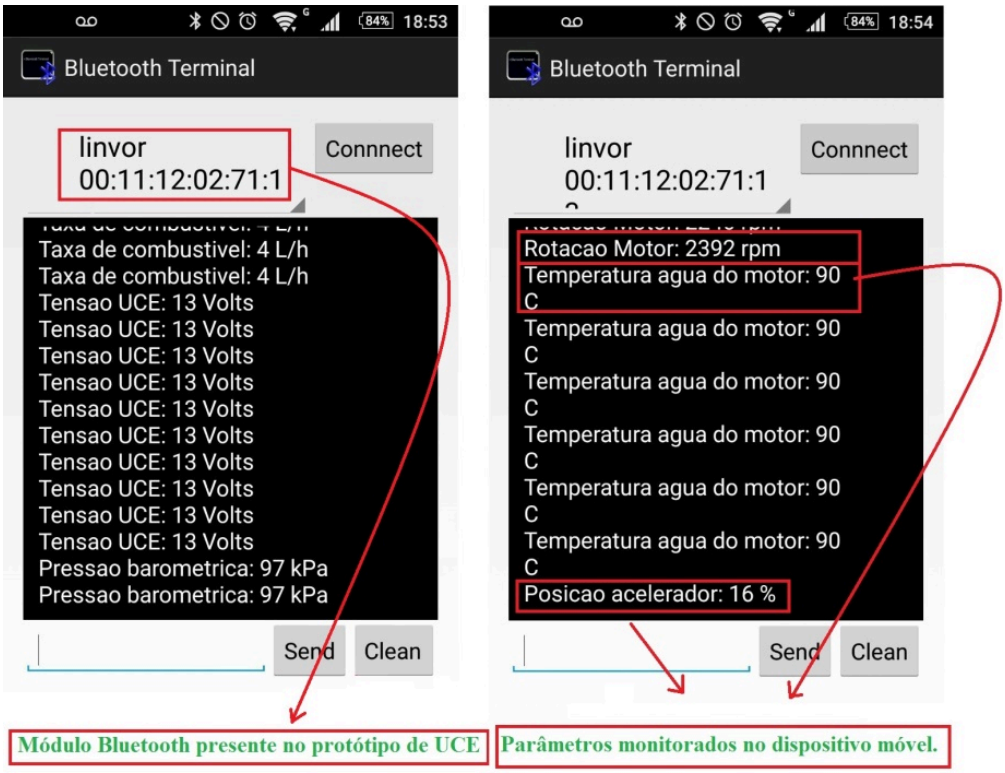
Fonte: Próprio autor.

5.9 Monitoramento *bluetooth*

Conforme já detalhado, objetivando expandir as possibilidades de implementação, foi adicionado ao protótipo de UCE um módulo *bluetooth* para comunicação sem fio. Apesar de a comunicação poder ser estabelecida entre o protótipo e microcomputadores (equipados com módulos *bluetooth* e *software* serial), o objetivo maior é para utilização em pareamento com dispositivos móveis *smartphones* e *tablets* devido à flexibilidade de uso. Tais dispositivos são normalmente equipados com sistemas de conexão à internet (GSM 3G/4G, *Wi-Fi*, etc.) e assim gerar interação entre o veículo e a rede global de computadores. A figura 46 demonstra o

uso do aplicativo *Bluetooth Terminal* (Sistema *Android*) comunicando um dispositivo *smartphone* ao veículo e obtendo informações sobre os parâmetros selecionados.

Figura 46 – Medição de parâmetros via *Bluetooth* utilizando smartphone.



Fonte: Próprio autor.

6 CONSIDERAÇÕES FINAIS

O trabalho proposto abordou um breve histórico sobre a evolução dos veículos automotores e a significativa participação global em pesquisas voltadas ao desenvolvimento e inovação destes, tendo como principal coadjuvante para obtenção de resultados, que revolucionaram a época, a aplicação de tecnologias embarcadas baseadas em microeletrônica por meio de centrais de comando eletrônico (UCE) aliadas aos diversos tipos de sensores e atuadores disponíveis no mercado. Em seguida foi apresentada a plataforma Arduino como sugestão de baixo custo para utilização como unidade de entrada, processamento e saída devido as suas excelentes características de *hardware* já detalhadas. O protocolo CAN por sua vez revelou-se determinante para a elaboração deste trabalho uma vez que é utilizado em escala mundial na indústria automobilística e deter seu conhecimento tornou-se de essencial importância para os engenheiros que venham a trabalhar no desenvolvimento de tecnologias automotivas conectadas em rede. O protocolo de diagnóstico OBDII, utilizado através do controlador CAN ELM327, mostrou ser a ponte entre a UCE principal do veículo de testes e o protótipo de UCE desenvolvido onde foi possível enviar requisições à UCE principal e utilizar informações recebidas na implementação de recursos a serem disponibilizados ao usuário.

O código fonte (algoritmo) do *firmware* implementado e utilizado no protótipo está disponível no apêndice deste trabalho e poderá ser utilizado, todo ou em parte, desde que haja a citação das fontes, pois foram utilizadas bibliotecas e trechos de códigos de terceiros para dar celeridade ao desenvolvimento do protótipo. A disponibilização do código foi motivada pelo desejo de disseminação do conhecimento, para que outros possam a partir deste desenvolver novas aplicações visando o bem comum da humanidade pelo progresso da ciência.

Os componentes e materiais utilizados são descritos ao longo do trabalho e a reprodução total ou parcial é livre.

Entre outros objetivos, a elaboração deste trabalho está vinculada ao desejo de inserir a Universidade Federal Rural do Semi-Árido (UFERSA) no contexto global de desenvolvimento de novas tecnologias a serem incorporadas à indústria automobilística. Isto se consuma com a finalização do trabalho em si, tornando-o material de apoio para uso dos docentes e discentes, nos pilares de ensino, pesquisa e extensão.

Entre os projetos já em andamento na UFERSA (Campus Caraúbas) há o de pesquisa, através do Car_ROS (curso de Engenharia Elétrica) e o de extensão, através do CARAUBAJA (curso de Engenharia Mecânica) regido pelo desafio Baja SAE BRASIL. Busca-se a integração entre ambos os projetos onde soluções diversas em Engenharia Elétrica (principalmen-

te nas áreas de Controle e Automação) sejam desenvolvidas e aplicadas ao veículo do projeto CARAUBAJA, integrando as engenharias ao enfrentar desafios reais em uma competição de nível global.

7 SUGESTÕES PARA TRABALHOS FUTUROS

Baseados nas informações descritas por este presente trabalho são listados abaixo algumas sugestões para desenvolvimento de trabalhos futuros, são elas:

- Desenvolvimento de aplicativo para sistema *Android* ou iOS que gerencie a atividade dos sensores;
- Desenvolvimento de recurso de gravação de log com informações obtidas dos sensores durante o percurso do veículo (velocidade, carga, consumo, tempo desde ignição, temperatura do sistema de arrefecimento, etc.);
- Desenvolvimento de recurso que monitore a temperatura de arrefecimento e alerte o piloto, por sinal sonoro, caso esta se eleve além do normal;
- Desenvolvimento de recurso que monitore a velocidade do veículo e atue a aceleração, mantendo a velocidade constante durante realização de percurso;
- Desenvolvimento de recurso que armazene em servidores *WEB* as informações do veículo referentes aos trajetos realizados e situação do veiculo, alertando o piloto sobre possíveis códigos de falhas apresentados;
- Desenvolvimento de recurso que incluam sensores GPS e acelerômetro para em conjunto com as informações relativas à velocidade do veículo identificar frenagens bruscas e colisões;
- Substituição do controlador CAN ELM 327, optando pelo modelo já embarcado com dispositivos de comunicação sem fio (*Bluetooth* ou *Wi-Fi*), eliminando assim o cabo de ligação entre o controlador e a UCE.

REFERÊNCIAS

ANJOS, Eduardo Giovannetti Pereira dos. **A evolução da eletrônica embarcada na indústria automobilística brasileira.** 2011. 125 f. Monografia (Especialização) - Curso de Engenharia de Processos Industriais, Instituto Mauá de Tecnologia, São Caetano do Sul, 2011.

ARDUINO. 2016. Disponível em: <https://www.hackaday.com>. Acesso em: 15 mar. 2016.

AUTOMOTIVE BUSINESS. São Paulo - Sp: Ab Digital, v. 1, n. 2, mar. 2010. Bimestral. Eletrônica Veicular: Você Vai Se Conectar. Disponível em: <https://issuu.com/automotivebusiness/docs/revista_automotive_business_2?mode=embed&layout=http://skin.issuu.com/v/light/layout.xml&showFlipBtn=true&logo=http://www.automotivebusiness.com.br/imagens/ab.gif>. Acesso em: 10 fev. 2016.

BARBOSA, Luiz Roberto Guimarães**. Redes CAN. 2003**. 14 f. Curso de Engenharia Elétrica, Universidade Federal de Minas Gerais, Belo Horizonte, 2003.

BARRAMENTO CAN. 2016. Disponível em: <http://www.machineworkshop.com.br>. Acesso em: 15 mar. 2016.

BOSCH, R. **Manual de Tecnologia Automotiva.** Tradução de Euryale de Jesus Zerbini et al. 25. Ed. São Paulo: Edgard Blücher, 2005. 1232 p.

BRASÍLIA. DENATRAN. (Ed.). **RESOLUÇÃO Nº 311, DE 03 DE ABRIL DE 2009.** 2009. Disponível em: <http://www.denatran.gov.br/download/resolucoes/resolucao_contran_311_09.pdf>. Acesso em: 20 abr. 2016.

CENTRAL de comando eletrônico. 2016. Disponível em: <http://www.bosch.com>. Acesso em: 29 mar. 2016.
COMUNICAÇÃO de Tempo Real em uma CAN. 2016. Disponível em: < http://www.cin.ufpe.br>. Acesso em: 10 mar. 2016.

ELETRÔNICA embarcada: Evolução dos sistemas elétricos/eletrônicos. 2016. Disponível em: <http://www.formula.ufscar.br/>. Acesso em: 29 mar. 2016.

ELM 327. 2016. Disponível em: <http://elmelectronics.com/DSheets/ELM327DS.pdf>. Acesso em: 15 mar. 2016.

EVOLUÇÃO da eletrônica e aumento da complexidade no diagnóstico de um veículo automotor. 2016. Disponível em: <http://www.noticiasdaoficinavw.com.br/v2/>. Acesso em: 29 mar. 2016.

GERALDO, Alessandre Rodrigues. **Rede CAN para acionamento e controle da plataforma autônoma para coleta de dados hidrológicos.** 2008. 64 f. TCC (Graduação) - Curso de Engenharia Elétrica, Universidade de São Carlos, São Carlos, 2008.

HACK an ELM327 Cable to make an Arduino OBD2 Scanner. 2016. Disponível em: <http://www.instructables.com>. Acesso em: 18 fev. 2016.

HILUX ano 2009. 2016. Disponível em: <https://www.google.com.br/imghp?hl=pt-PT>. Acesso em: 16 mar. 2016.

KAUARK, Fabiana; MANHÃES, Fernanda; MEDEIROS, Carlos. **Metodologia da Pesquisa: guia prático.** Itabuna: Via Litterarum, 2010.

LED cube. 2016. Disponível em: < http://www.instructables.com/id/Led-Cube-8x8x8/>. Acesso em: 5 mar. 2016.

MARQUES, Marco Antonio. **CAN Automotivo Sistema de Monitoramento.** 2004. 163 f. Dissertação (Mestrado) - Curso de Engenharia Elétrica, Universidade Federal de Itajubá, Itajubá, 2004.

MAURICI, Alessandro Barreiros. **Suportes a redes CAN para aplicações embarcadas.** 2005. 93 f. TCC (Graduação) - Curso de Ciência da Computação, Informática e Estatística, Universidade Federal de Santa Catarina, Florianópolis, 2005.

MICROCONTROLADOR 18f2580. 2016. Disponível em: <https://www.waveshare.com>. Acesso em: 18 fev. 2016.

MÓDULO bluetooth. 2016. Disponível em: <https://www.arduinoecia.com>. Acesso em: 15 mar. 2016.

MÓDULO buzzer. 2016. Disponível em: <http://www.mercadolivre.com>. Acesso em: 15 mar. 2016.

MÓDULO display de led´s. 2016. Disponível em: <http://www.electroschematics.com>. Acesso em: 15 mar. 2016.

O QUE é rede CAN. 2016. Disponível em: <http://machineworkshop.com.br/tag/o-que-e-rede-can/>. Acesso em: 15 abr. 2016.

OBD-II PIDs. 2016. Disponível em: <https://en.wikipedia.org/wiki/OBD-II_PIDs>. Acesso em: 15 mar. 2016.

OBDII. 2016. Disponível em: <http://www.play.google.com>. Acesso em: 15 mar. 2016.

ON board diagnostics. 2016. Disponível em: <https://en.wikipedia.org/wiki/On-board_diagnostics>. Acesso em: 15 mar. 2016.

PLATAFORMA Arduino. 2016. Disponível em: <https://www.arduino.cc/>. Acesso em: 15 fev. 2016.

PROJETO arducopter. 2016. Disponível em: < http://diydrones.com >. Acesso em: 10 mar. 2016.

PROTOCOLO CAN. 2016. Disponível em: < http://machineworkshop.com.br >. Acesso em: 20 fev. 2016.

REPOSITÓRIO. 2016. Disponível em: <http://www.code.google.com>. Acesso em: 29 mar. 2016.

SAKAI, R. M. R. **Rede Serial para comunicação de Dados e Controle em Sistema Embarcado:** Estudo de implementação da ISO 11783. 2008. 119f. Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2008.

SHIELD lcd. 2016. Disponível em: <http://www.flipflop.com>. Acesso em: 20 mar. 2016.

VOLKSWAGEN-AUDI. **Data Exchange On The CAN Bus I.** Self-Study Programme 238. Disponível em: <http://www.volkspage.net/technik/ssp/ssp/SSP_238.pdf>. Acesso em: 10 abr. 2016.

APÊNDICE

```
/*****************************************************************************************************************************************************
*           Universidade Federal Rural do Semi-Arido - UFERSA                                                                                        *
*         Trabalho de conclusao do curso de Engenharia eletrica                                                                                      *
*                     Campus Caraubas-RN                                                                                                             *
*                  Aluno: Bemielison G. S. Bezerra                                                                                                   *
*  Utilizados trechos de codigos e/ou bibliotecas dos links:                                                                                         *
*  http://www.instructables.com/id/Hack-an-ELM327-Cable-to-make-an-Arduino-OBD2-Scann/?ALLSTEPS em 19/05/2016;                                        *
*  https://github.com/wayoda/LedControl                                                                                                              *
*  http://playground.arduino.cc/Main/LiquidCrystal                                                                                                   *
*                                                                                                                                                    *
*                                                                                                                                                    *
*  OBSERVACOES:                                                                                                                                      *
*  Utilizaçao do modelo Arduino Mega;                                                                                                                *
*  Inserida biblioteca LiquidCrystal.h;                                                                                                              *
*  Corrigidas as formulas para calculo dos parametros recebidos do ELM327 (codigo exemplo);                                                         *
*  Acrescentados novos parametros de diagnostico em relaçao ao codigo exemplo;                                                                       *
*  Serial1 como sendo a comunicacao com o ELM327 e Serial2 para comunicacao bluetooth;                                                               *
*  Serial padrao reservada para conexao ao pc;                                                                                                       *
*  Matrix de led para exibicao do range;                                                                                                             *
*  Retirada do codigo exemplo a conversao de string para long por acarretar erros na conversao;                                                      *
*  Buzzer para alarme de acesso de velocidade do veiculo.                                                                                            *
* ***************************************************************************************************************************************************/


#include <LiquidCrystal.h>
#include "LedControl.h"                       //matrix

  const int buzzer = 28;                      //pino buzzer 28
  LedControl lc=LedControl(22,24,26,1);       //matrix  Pin 22: DIN; 24: CLK; 26 CS. MAX 7219
  unsigned long delaytime=30;                 //matrix

  int CmdCount=1;     byte inData;              char inChar;              String BuildINString="";   String DisplayString="";
  long DisplayValue;  String SentMessage="";    int ByteCount=0;          long A;                    int B;
  int WorkingVal;     String WorkingString="";  String WorkingStringA=""; String WorkingStringB=""; int x;   int z=0;

  byte N1[8]={B10000000}; byte N2[8]={B10000001}; byte N3[8]={B10000011}; byte N4[8]={B10000111};
  byte N5[8]={B10001111}; byte N6[8]={B10011111}; byte N7[8]={B10111111}; byte N8[8]={B11111111};

  String A_1;  String A_2;  String A_3;  String A_4;

  int i=0;         int Valor1 = 0;          int Valor2 = 0;          int Valor3 = 0;
  int Valor4 = 0;  int Valor_parcial1 = 0;  int Valor_parcial2 = 0;  int Valor_total = 0;


  int MenuID=0;  //Declaracao dos botoes de acesso ao menu

LiquidCrystal lcd(8, 9, 4, 5, 6, 7); //definicao dos pinos de conexao ao display


void setup() {


  lc.shutdown(0,false);            // funcao para despertar o controlador
  lc.setIntensity(0,8);            // ajuste de brilho range de 0 a 15
  lc.clearDisplay(0);              // limpar o display

  lcd.begin(16, 2);     // Configurando display.
  lcd.setCursor(0, 0);  // Print a message to the LCD.
  Bootup();             //Iniciando...
  Retry:
  lcd.setCursor(0, 0);  lcd.print("Conectando......");  lcd.setCursor(0, 1);  lcd.print("               ");
  Serial1.begin(38400);  delay(500);  Serial.begin(38400);  delay(500); Serial2.begin(9600);  delay(500);
  delay (5000);
  SentMessage = "ATI";    //Mensagem a ser enviada para o controlador CAN.
  Serial1.println("ATI"); // Mensagem enviada para o controlador CAN
  delay(500);
  ReadData();             // Funcao para leitura serial dos dados vindos do controlador CAN

  if (BuildINString.substring(1,7)=="ELM327"){
      lcd.setCursor(0, 0);   lcd.print("Bem Vindo!      ");      Serial.print("Bem Vindo! \n" ); Serial2.print("Bem Vindo! \n" );  bip();
      lcd.setCursor(0, 1);   lcd.print("Conectado!      ");   Serial.print("Conectado! \n" ); Serial2.print("Conectado! \n" );
      delay(1500);
  }else {
        lcd.setCursor(0, 0);   lcd.print("Erro!           ");      Serial.print("Erro!           \n" );   bip();
        lcd.setCursor(0, 1);   lcd.print("Falha Conexao...");      Serial.print("Falha Conexao... \n" );
        Serial2.print("Falha Conexao... \n" );
        delay(1500);      goto Retry;
        }
  delay(1500);


  }
```

```
void loop() {

  configurar_display ();  parametros_leitura ();

 }

  void ReadData() {    //Funçao para leitura serial e execuçao de açoes.
    // criado vetor com 8 saidas, onde 1 representa led aceso e 0 apagado
    BuildINString="";
    while(Serial1.available() > 0) {
        inData=0; inChar=0;  inData = Serial1.read();   inChar=char(inData);  BuildINString = BuildINString + inChar;}
    BuildINString.replace(SentMessage,"");  BuildINString.replace(">","");  BuildINString.replace("OK","");
    BuildINString.replace("STOPPED","");  BuildINString.replace("SEARCHING","");  BuildINString.replace("NO DATA","");
    BuildINString.replace("?","");  BuildINString.replace(",","");
    //Leitura dos codigos recebidos conforme mensagem enviada
   if (SentMessage=="01 0C")      // recebe o valor vindo do controlador CAN e calcula a RPM (0 a 7000, no caso da hilux)
   {
     ler_dado_AB ();
     DisplayValue = (((Valor_parcial1*256)+Valor_parcial2)/4);             //calculo para obtencao da grandeza desejada
     matriz_rpm ();
     Serial.print("Valor obtido parcial 1: ");    Serial.print(Valor_parcial1);    Serial.print("\n");
     Serial.print("Valor obtido parcial 2: "); Serial.print(Valor_parcial2);    Serial.print("\n");
     Serial.print("Valor calculado: ");    Serial.print(DisplayValue);    Serial.print("\n");
     DisplayString = String(DisplayValue) + " rpm            ";
     lcd.setCursor(0, 1);   lcd.print(DisplayString);   Serial.print(DisplayString);   Serial.print(" RPM \n");
     Serial2.print("Rotacao Motor: "); Serial2.print(DisplayString);   Serial2.print("\n");

     if (DisplayValue > 4000){ bip();};
     }

    if (SentMessage=="01 42")      // Tensao da ECU (0 a 65.535)
   {
     ler_dado_AB ();
     DisplayValue = (((Valor_parcial1*256)+Valor_parcial2)/1000);             //calculo para obtencao da grandeza desejada
     matriz_tensao_UCE ();

    Serial.print("Valor obtido parcial 1: ");    Serial.print(Valor_parcial1);    Serial.print("\n");
    Serial.print("Valor obtido parcial 2: "); Serial.print(Valor_parcial2);    Serial.print("\n");
    Serial.print("Valor calculado: ");    Serial.print(DisplayValue);    Serial.print("\n");
    DisplayString = String(DisplayValue) + " Volts           ";
    lcd.setCursor(0, 1);   lcd.print(DisplayString);   Serial.print(DisplayString);   Serial.print(" Volts \n");
    Serial2.print("Tensao UCE: "); Serial2.print(DisplayString);   Serial2.print("\n");
    if (DisplayValue > 15){ bip();};
   }

  if (SentMessage=="01 5E")      //  Taxa combustivel do motor (0 a 3.212,75)
   {
     ler_dado_AB ();
     DisplayValue = (((Valor_parcial1*256)+Valor_parcial2)/20);            //calculo para obtencao da grandeza desejada
     matriz_acel_carga ();
     Serial.print("Valor obtido parcial 1: ");    Serial.print(Valor_parcial1);    Serial.print("\n"); Serial.print("Valor obtido parcial 2: ");
     Serial.print(Valor_parcial2);    Serial.print("\n");  Serial.print("Valor calculado: ");    Serial.print(DisplayValue);
     Serial.print("\n");     DisplayString = String(DisplayValue) + " L/h           ";
     lcd.setCursor(0, 1);   lcd.print(DisplayString);   Serial.print(DisplayString);   Serial.print(" L/h \n");
     Serial2.print("Taxa de combustivel: "); Serial2.print(DisplayString);  Serial2.print("\n");
    }

   if (SentMessage=="01 0D")      // calculo da velocidade do veiculo  (0 a 255)
   {
    ler_dado_A ();
    DisplayValue = Valor_parcial1;
    matriz_vel_temp ();
    Serial.print("Valor obtido: ");    Serial.print(Valor_parcial1);    Serial.print("\n");    Serial.print("Valor calculado: ");
    Serial.print(DisplayValue);    Serial.print("\n");
    DisplayString = String(DisplayValue) + " km/h           ";   lcd.setCursor(0, 1);   lcd.print(DisplayString);
    Serial.print(DisplayString);   Serial.print(" km/h \n"); Serial2.print("Velocidade Veiculo: "); Serial2.print(DisplayString);  Serial2.print("\n");
    if (DisplayValue > 80){ bip();};  // Se velocidade acima de 80km, soar bip.
    }
   if (SentMessage=="01 05")      // temperatura da agua do motor (-40 a 215)
   {
```

```
  ler_dado_A ();
  DisplayValue = (Valor_parcial1 - 40);
  matriz_vel_temp ();
  Serial.print("Valor obtido: ");    Serial.print(Valor_parcial1);    Serial.print("\n");    Serial.print("Valor calculado: ");
  Serial.print(DisplayValue);    Serial.print("\n");
  DisplayString = String(DisplayValue) + " C              ";    lcd.setCursor(0, 1);    lcd.print(DisplayString);
  Serial.print(DisplayString);    Serial.print(" Celsios \n"); Serial2.print("Temperatura agua do motor: ");
  Serial2.print(DisplayString);  Serial2.print("\n");
  if (DisplayValue > 85){ bip();};
 }
 if (SentMessage=="01 11")  {  // posicao do acelerador  (0 a 100)
  ler_dado_A ();
  DisplayValue = (Valor_parcial1/2.55);
  matriz_acel_carga ();
  Serial.print("Valor obtido: ");    Serial.print(Valor_parcial1);    Serial.print("\n");    Serial.print("Valor calculado: ");
  Serial.print(DisplayValue);    Serial.print("\n");
  DisplayString = String(DisplayValue) + " %              ";   lcd.setCursor(0, 1);   lcd.print(DisplayString);
  Serial.print(DisplayString);   Serial.print(" % \n"); Serial2.print("Posicao acelerador: "); Serial2.print(DisplayString);
  Serial2.print("\n");
 }
 if (SentMessage=="01 04")  {  // carga do motor  (0 a 100)
  ler_dado_A ();
  DisplayValue = (Valor_parcial1/2.55);
  matriz_acel_carga ();
  Serial.print("Valor obtido: ");    Serial.print(Valor_parcial1);    Serial.print("\n");    Serial.print("Valor calculado: ");
  Serial.print(DisplayValue);    Serial.print("\n");
  DisplayString = String(DisplayValue) + " %              ";   lcd.setCursor(0, 1);   lcd.print(DisplayString);
  Serial.print(DisplayString);   Serial.print(" % \n"); Serial2.print("Carga do motor: "); Serial2.print(DisplayString);
  Serial2.print("\n");
  if (DisplayValue > 90){ bip();};
 }


  if (SentMessage=="01 33")  {  // Pressao barometrica absoluta  (0 a 255)
   ler_dado_A ();
   DisplayValue = Valor_parcial1;
   matriz_vel_temp ();
   Serial.print("Valor obtido: ");    Serial.print(Valor_parcial1);    Serial.print("\n");    Serial.print("Valor calculado: ");
   Serial.print(DisplayValue);    Serial.print("\n");
   DisplayString = String(DisplayValue) + " kPa             ";   lcd.setCursor(0, 1);   lcd.print(DisplayString);
   Serial.print(DisplayString);   Serial.print(" kPa \n"); Serial2.print("Pressao barometrica: "); Serial2.print(DisplayString);
   Serial2.print("\n");
  }
 }
```

```
void Bootup() {        // Funcao de inicializacao
 lcd.print("ENG. ELET.UFERSA");
     for (int i=0; i <= 4; i++) {
         for (int j=1; j <= 3; j++){
         if(j==1){lcd.setCursor(0, 1);lcd.print ("-");delay(200);}
         if(j==2){lcd.setCursor(0, 1);lcd.print ("/");delay(200);}
         if(j==3){lcd.setCursor(0, 1);lcd.print ("|");delay(200);}}}
    delay(1000);
}
```

```
void substituto () {
   Valor1 = 0; Valor2 = 0; Valor_parcial1 = 0;
   A_1 = WorkingString.substring(0,1);  Serial.print("Valor A_1 primeira posicao: "); Serial.print(A_1);    Serial.print("\n");
   A_2 = WorkingString.substring(1,2);  Serial.print("Valor A_2 segunda posicao: ");  Serial.print(A_2);    Serial.print("\n");

     if (A_1 == "0"){ Valor1 = 0;}
         else if (A_1 == "1"){    Valor1 = 16; }else if (A_1 == "2"){    Valor1 = 32; }else if (A_1 == "3"){    Valor1 = 48;   }
         else if (A_1 == "4"){    Valor1 = 64; }else if (A_1 == "5"){    Valor1 = 80; }else if (A_1 == "6"){    Valor1 = 96;   }
         else if (A_1 == "7"){    Valor1 = 112;}else if (A_1 == "8"){    Valor1 = 128;}else if (A_1 == "9"){    Valor1 = 144;  }
         else if (A_1 == "A"){    Valor1 = 160;}else if (A_1 == "B"){    Valor1 = 176;}else if (A_1 == "C"){    Valor1 = 192;  }
         else if (A_1 == "D"){    Valor1 = 208;}else if (A_1 == "E"){    Valor1 = 224;}else if (A_1 == "F"){    Valor1 = 240;  }

     if (A_2 == "0"){  Valor2 = 0;}
         else if (A_2 == "1"){    Valor2 = 1; }else if (A_2 == "2"){    Valor2 = 2;   }else if (A_2 == "3"){    Valor2 = 3;    }
         else if (A_2 == "4"){    Valor2 = 4; }else if (A_2 == "5"){    Valor2 = 5;   }else if (A_2 == "6"){    Valor2 = 6;    }
         else if (A_2 == "7"){    Valor2 = 7; }else if (A_2 == "8"){    Valor2 = 8;   }else if (A_2 == "9"){    Valor2 = 9;    }
         else if (A_2 == "A"){    Valor2 = 10;}else if (A_2 == "B"){    Valor2 = 11;  }else if (A_2 == "C"){    Valor2 = 12;   }
         else if (A_2 == "D"){    Valor2 = 13;}else if (A_2 == "E"){    Valor2 = 14;  }else if (A_2 == "F"){    Valor2 = 15;   }
   A_3 = WorkingString.substring(2,3);  Serial.print("Valor A_3 terceira posicao: "); Serial.print(A_3);    Serial.print("\n");
   A_4 = WorkingString.substring(3,4);  Serial.print("Valor A_4 quarta posicao: ");   Serial.print(A_4);    Serial.print("\n");
```

```
  if (A_3 != ""  &&  A_4 =="") {
       if (A_3 == "0"){  Valor3 = 0;}
         else if (A_3 == "1"){    Valor3 = 1; }else if (A_3 == "2"){    Valor3 = 2;  }else if (A_3 == "3"){    Valor3 = 3;   }
         else if (A_3 == "4"){    Valor3 = 4; }else if (A_3 == "5"){    Valor3 = 5;  }else if (A_3 == "6"){    Valor3 = 6;   }
         else if (A_3 == "7"){    Valor3 = 7; }else if (A_3 == "8"){    Valor3 = 8;  }else if (A_3 == "9"){    Valor3 = 9;   }
         else if (A_3 == "A"){    Valor3 = 10;}else if (A_3 == "B"){    Valor3 = 11; }else if (A_3 == "C"){    Valor3 = 12;  }
         else if (A_3 == "D"){    Valor3 = 13;}else if (A_3 == "E"){    Valor3 = 14; }else if (A_3 == "F"){    Valor3 = 15;  }}
  if (A_4 !=""){
       if (A_3 == "0"){  Valor3 = 0;}
         else if (A_3 == "1"){    Valor3 = 16; }else if (A_3 == "2"){    Valor3 = 32; }else if (A_3 == "3"){    Valor3 = 48;   }
         else if (A_3 == "4"){    Valor3 = 64; }else if (A_3 == "5"){    Valor3 = 80; }else if (A_3 == "6"){    Valor3 = 96;   }
         else if (A_3 == "7"){    Valor3 = 112;}else if (A_3 == "8"){    Valor3 = 128;}else if (A_3 == "9"){    Valor3 = 144;  }
         else if (A_3 == "A"){    Valor3 = 160;}else if (A_3 == "B"){    Valor3 = 176;}else if (A_3 == "C"){    Valor3 = 192;  }
         else if (A_3 == "D"){    Valor3 = 208;}else if (A_3 == "E"){    Valor3 = 224;}else if (A_3 == "F"){    Valor3 = 240;  }
      if (A_4 == "0"){  Valor4 = 0;}
         else if (A_4 == "1"){    Valor4 = 1; }else if (A_4 == "2"){    Valor4 = 2;   }else if (A_4 == "3"){    Valor4 = 3;    }
         else if (A_4 == "4"){    Valor4 = 4; }else if (A_4 == "5"){    Valor4 = 5;   }else if (A_4 == "6"){    Valor4 = 6;    }
         else if (A_4 == "7"){    Valor4 = 7; }else if (A_4 == "8"){    Valor4 = 8;   }else if (A_4 == "9"){    Valor4 = 9;    }
         else if (A_4 == "A"){    Valor4 = 10;}else if (A_4 == "B"){    Valor4 = 11;  }else if (A_4 == "C"){    Valor4 = 12;   }
         else if (A_4 == "D"){    Valor4 = 13;}else if (A_4 == "E"){    Valor4 = 14;  }else if (A_4 == "F"){    Valor4 = 15;   }}

   Serial.print("Valor1:  "); Serial.print(Valor1);    Serial.print("\n");    Serial.print("Valor2:  "); Serial.print(Valor2);
   Serial.print("\n");    Serial.print("Valor3:  "); Serial.print(Valor3);    Serial.print("\n");    Serial.print("Valor4:  ");
   Serial.print(Valor4);    Serial.print("\n");

   Valor_parcial1 = Valor1 + Valor2; Valor_parcial2 = Valor3 + Valor4;

   Serial.print("Valor parcial 1:  "); Serial.print(Valor_parcial1);    Serial.print("\n"); Serial.print("Valor parcial 2:  ");
   Serial.print(Valor_parcial2);    Serial.print("\n");
 }


void matriz_rpm () {
 x = DisplayValue;
  //RPM
       if (x < 500)          {nivel0(); }
   else if (x > 500  && x < 1000) {nivel1(); }    else if (x > 1000 && x < 1500) {nivel2(); }    else if (x > 1500 && x < 2000) {nivel3(); }
   else if (x > 2000 && x < 2500) {nivel4(); }    else if (x > 2500 && x < 3500) {nivel5(); }    else if (x > 3500 && x < 4500) {nivel6(); }
   else if (x > 4500)             {nivel7(); }    delay(delaytime);   }

void matriz_vel_temp () {
  //velocidade e temperatura
  x = DisplayValue;
       if (x < 25)             {nivel0(); }
   else if (x > 25 && x < 50)   {nivel1(); }      else if (x > 50 && x < 75)   {nivel2(); }      else if (x > 75 && x < 100)  {nivel3(); }
   else if (x > 100 && x < 125) {nivel4(); }      else if (x > 125 && x < 150) {nivel5(); }      else if (x > 150 && x < 175) {nivel6(); }
   else if (x > 175)            {nivel7(); }      delay(delaytime);   }

void matriz_acel_carga () {
  //acelerador e carga do motor
  x = DisplayValue;
       if (x < 5)              {nivel0(); }
   else if (x > 5  && x < 20)   {nivel1(); }      else if (x > 20 && x < 35)   {nivel2(); }      else if (x > 35 && x < 45)   {nivel3(); }
   else if (x > 45 && x < 60)   {nivel4(); }      else if (x > 60 && x < 75)   {nivel5(); }      else if (x > 75 && x < 90)   {nivel6(); }
   else if (x > 90 && x < 100)  {nivel7(); }      delay(delaytime);   }

void matriz_tensao_UCE () {
  //tensao UCE
  x = DisplayValue;
       if (x < 5)              {nivel0(); }
   else if (x > 5  && x < 10)   {nivel1(); }      else if (x > 10 && x < 15)   {nivel2(); }      else if (x > 15 && x < 20)   {nivel3(); }
   else if (x > 20 && x < 25)   {nivel4(); }      else if (x > 25 && x < 30)   {nivel5(); }      else if (x > 30 && x < 35)   {nivel6(); }
   else if (x > 35 )  {nivel7(); }               delay(delaytime);   }
```

```
void ler_dado_A () {


  WorkingString = BuildINString.substring(7,9);  Serial.print ("mensagem recebida Workstring 7,9: ");    Serial.print (WorkingString);
  Serial.print("\n");
  substituto();
  }


void ler_dado_AB () {

    WorkingStringA = BuildINString.substring(7,9);   Serial.print ("mensagem recebida Workstring 7,9: ");    Serial.print (WorkingStringA);    Serial.print("\n");
    WorkingStringB = BuildINString.substring(11,13); Serial.print ("mensagem recebida Workstring 11,13: ");    Serial.print (WorkingStringB);    Serial.print("\n");
    WorkingString  = WorkingStringA;
    WorkingString += WorkingStringB;   Serial.print ("mensagem recebida Workstring A + B: ");    Serial.print (WorkingString);    Serial.print("\n");
    substituto();  //funcao que calcula o valor recebido pela string
}

void configurar_display () {
 int x;   x = analogRead (0);            // Leitura dos botoes do shield do display.
 lcd.setCursor(10,1);            //Menu de opçoes
 if (x > -10 and x < 100){if (MenuID < 11){MenuID++;}delay(250);} //Direito
 if (x > 100 and x < 200){ lcd.print ("Up      ");} //Alto
 if (x > 200 and x < 400){ lcd.print ("Down    ");} //Baixo
 if (x > 400 and x < 600){ if (MenuID > 0){MenuID--;}delay(250);}//Esquerdo
 if (x > 600 and x < 800){ lcd.print ("Select  ");} //Selecionar
}

void parametros_leitura () {
 if (MenuID==0){lcd.setCursor(0, 0);lcd.print("01 Carga Motor  ");lcd.setCursor(0, 1);lcd.print(DisplayString); WorkingString = ""; SentMessage = "01 04";
 Serial1.println("01 04"); Serial.print("Solicitada Carga do motor em %... \n" );delay(300);ReadData();}
 if (MenuID==1){lcd.setCursor(0, 0);lcd.print("02 Rotacao Motor");lcd.setCursor(0, 1);lcd.print(DisplayString); WorkingString = ""; SentMessage = "01 0C";
 Serial1.println("01 0C"); Serial.print("Solicitada Rotaçao do motor em RPM... \n" );delay(300);ReadData();}
 if (MenuID==2){lcd.setCursor(0, 0);lcd.print("03 Temp_Agua_Mot");lcd.setCursor(0, 1);lcd.print(DisplayString); WorkingString = ""; SentMessage = "01 05";
 Serial1.println("01 05"); Serial.print("Solicitada Temperatura da agua do radiador... \n" );delay(300);ReadData();}
 if (MenuID==3){lcd.setCursor(0, 0);lcd.print("04 Acelerador   ");lcd.setCursor(0, 1);lcd.print(DisplayString); WorkingString = ""; SentMessage = "01 11";
 Serial1.println("01 11"); Serial.print("Solicitada Posicao do acelerador em %... \n" );delay(300);ReadData();}
 if (MenuID==4){lcd.setCursor(0, 0);lcd.print("05 Velocidade   ");lcd.setCursor(0, 1);lcd.print(DisplayString); WorkingString = ""; SentMessage = "01 0D";
 Serial1.println("01 0D"); Serial.print("Solicitada Velocidade do veiculo em km/h... \n" );delay(300);ReadData();}
 if (MenuID==5){lcd.setCursor(0, 0);lcd.print("06 Pressao Barom");lcd.setCursor(0, 1);lcd.print(DisplayString); WorkingString = ""; SentMessage = "01 33";
 Serial1.println("01 33"); Serial.print("Solicitada Pressao Barometrica...\n" );delay(300);ReadData();}
 if (MenuID==6){lcd.setCursor(0, 0);lcd.print("07 Tensao UCE   ");lcd.setCursor(0, 1);lcd.print(DisplayString); WorkingString = ""; SentMessage = "01 42";
 Serial1.println("01 42"); Serial.print("Solicitado Tensao UCE...\n" );delay(300);ReadData();}
 if (MenuID==7){lcd.setCursor(0, 0);lcd.print("08 Taxa Comb Mot");lcd.setCursor(0, 1);lcd.print(DisplayString); WorkingString = ""; SentMessage = "01 5E";
 Serial1.println("01 5E"); Serial.print("Solicitado Taxa Combustivel...\n" );delay(300);ReadData();}
}
```

```
void nivel0(){
lc.setRow(0,0,N1[0]);  lc.setRow(0,1,N1[0]);  lc.setRow(0,2,N1[0]); lc.setRow(0,3,N1[0]);  lc.setRow(0,4,N1[0]);
lc.setRow(0,5,N1[0]);  lc.setRow(0,6,N1[0]);  lc.setRow(0,7,N1[0]);  delay(delaytime);  }

void nivel1(){
lc.setRow(0,0,N2[0]);  lc.setRow(0,1,N2[0]);  lc.setRow(0,2,N2[0]); lc.setRow(0,3,N2[0]);  lc.setRow(0,4,N2[0]);
lc.setRow(0,5,N2[0]);  lc.setRow(0,6,N2[0]);  lc.setRow(0,7,N2[0]);  delay(delaytime);  }

void nivel2(){
lc.setRow(0,0,N3[0]);  lc.setRow(0,1,N3[0]);  lc.setRow(0,2,N3[0]); lc.setRow(0,3,N3[0]);  lc.setRow(0,4,N3[0]);
lc.setRow(0,5,N3[0]);  lc.setRow(0,6,N3[0]);  lc.setRow(0,7,N3[0]);  delay(delaytime);  }

void nivel3(){
lc.setRow(0,0,N4[0]);  lc.setRow(0,1,N4[0]);  lc.setRow(0,2,N4[0]); lc.setRow(0,3,N4[0]);  lc.setRow(0,4,N4[0]);
lc.setRow(0,5,N4[0]);  lc.setRow(0,6,N4[0]);  lc.setRow(0,7,N4[0]);  delay(delaytime);  }

void nivel4(){
lc.setRow(0,0,N5[0]);  lc.setRow(0,1,N5[0]);  lc.setRow(0,2,N5[0]); lc.setRow(0,3,N5[0]);  lc.setRow(0,4,N5[0]);
lc.setRow(0,5,N5[0]);  lc.setRow(0,6,N5[0]);  lc.setRow(0,7,N5[0]);  delay(delaytime);  }

void nivel5(){
lc.setRow(0,0,N6[0]);  lc.setRow(0,1,N6[0]);  lc.setRow(0,2,N6[0]); lc.setRow(0,3,N6[0]);  lc.setRow(0,4,N6[0]);
lc.setRow(0,5,N6[0]);  lc.setRow(0,6,N6[0]);  lc.setRow(0,7,N6[0]);  delay(delaytime);  }

void nivel6(){
lc.setRow(0,0,N7[0]);  lc.setRow(0,1,N7[0]);  lc.setRow(0,2,N7[0]); lc.setRow(0,3,N7[0]);  lc.setRow(0,4,N7[0]);
lc.setRow(0,5,N7[0]);  lc.setRow(0,6,N7[0]);  lc.setRow(0,7,N7[0]);  delay(delaytime);  }

void nivel7(){
lc.setRow(0,0,N8[0]);  lc.setRow(0,1,N8[0]);  lc.setRow(0,2,N8[0]); lc.setRow(0,3,N8[0]);  lc.setRow(0,4,N8[0]);
lc.setRow(0,5,N8[0]);  lc.setRow(0,6,N8[0]);  lc.setRow(0,7,N8[0]);  delay(delaytime);  }


void bip (){

 pinMode(buzzer, OUTPUT);   delay(300);
 pinMode(buzzer, INPUT);    delay(100);
}
```