CENTRO UNIVERSITÁRIO DE BRASÍLIA

FACULDADE DE TECNOLOGIA E CIÊNCIAS SOCIAIS APLICADAS

CURSO DE ENGENHARIA DE COMPUTAÇÃO

SISTEMA DE MONITORAMENTO ELETRÔNICO AUTOMOTIVO

AUTOR(A): ÁLVARO SANTANA DOS SANTOS JÚNIOR

MONOGRAFIA DE CONCLUSÃO DO CURSO DE ENGENHARIA DE COMPUTAÇÃO

Orientador: Profª. MC. Maria Marony Sousa F. Nascimento

Brasília, 23 de novembro de 2009.

ÁLVARO SANTANA DOS SANTOS JÚNIOR

SISTEMA DE MONITORAMENTO ELETRÔNICO AUTOMOTIVO

Trabalho de conclusão do curso de Engenharia de Computação, da Faculdade de Tecnologia e Ciências Sociais Aplicadas do Centro Universitário de Brasília - UniCEUB.

Profª Orientadora: MC. Maria Marony Sousa F. Nascimento

Brasília, 2009

AGRADECIMENTOS

Agradeço a todos que contribuíram para a conclusão deste projeto, principalmente a minha família pela compreensão e apoio dados ao longo desse semestre.

Não poderia deixar de agradecer ao corpo docente do curso de Engenharia de Computação do UniCEUB, que foi fundamental para o aprendizado necessário para a implementação deste projeto. Em especial agradeço a minha orientadora, a professora Maria Marony, pelo apoio, dedicação e contribuição para a melhoria e aperfeiçoamento do trabalho aqui apresentado.

Aos meus colegas de turma agradeço pelo apoio e motivação, principalmente nas dificuldades. Agradeço em especial os colegas Diego, Leandro, Marcus Vinícius, Robson e Tiago. Agradeço também aos meus amigos Paulo, Diogo, Davi e Marcus Túlio que me auxiliaram nos testes com o veículo em movimento.

Por fim, agradeço também a equipe do SENAI-DF, representada pelos instrutores Lester e Franco pelas informações prestadas. Agradeço a equipe da Taguauto pelo atendimento cortês e atencioso.

RESUMO

Neste trabalho é apresentado o protótipo de um sistema de monitoramento eletrônico automotivo. Neste protótipo, são mostradas ao condutor do veículo, informações de condução e das condições do sistema de gerenciamento do motor do veículo, com o objetivo de suprir a falta das informações disponibilizadas no painel de instrumentos, sobretudo em veículos populares.

Este protótipo coleta os dados do sistema de injeção eletrônica do veículo através de uma tomada de diagnóstico, processa essas informações, e as disponibiliza ao condutor do veículo através de um dispositivo móvel, minimizando a perca de dirigibilidade do veículo.

Palavras-chave: Monitoramento automotivo, Key Word 1281, diagnose on-board II (OBDII), computador de bordo.

ABSTRACT

This Project presents the prototype of a system of electronic monitoring automotive. In this prototype, are showed to the vehicle's driver, information of driving and of the conditions of the management system of the vehicle's engine, in order to compensate for the lack of information available on the dashboard, especially in popular vehicles.

This prototype collects data from electronic injection system of the vehicle through a connector of diagnostic, processes this information, and makes them available to the driver through a mobile device, minimizing the loss of steerability.

Keywords: Monitoring automotive, Key Word 1281, on-board diagnostics II (OBDII), board computer.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE EQUAÇÕES

LISTA DE SÍMBOLOS E ABREVIATURAS

ASCII: American Standart code for Information

AT: Alto Torque

CAN: Controller Area Network

CARB: California Air Resource Board

CI: Compression Ignition

CI: Circuito Integrado

CONAMA: Conselho Nacional do Meio Ambiente

CTN: Coeficiente de Temperatura Negativo

CTP: Coeficiente de Temperatura Positivo

EGAS: Acelerador Eletrônico

EPA: Federal Environmental Protection Agency

EPROM: Erasable Programmable Read Only Memory

ETX: End of Text

FTDI: Future Technology Devices International Limited

GND: Ground

GSM: Global System for Mobile

IEEE: Instituto de Engenheiros Eletricistas e Eletrônicos

ISO: International Organization for Standardization

KW1281: Key Word 1281

MSB: Most Significant Bit

OBD II: On Board Diagnosis II

PMS: Ponto morto superior

PWM: Pulse Width Modulation

RPM: Rotação (ões) Por Minuto

SAE: Society of Automotive Engineers

SI: Spark Ignition

SID: Service Identification

SIM: Security Identify Module

TDC: Top Dead Centre,

UCE: Unidade de Controle Eletrônico

USB: Universal Serial Bus

VAG: Volkswagen Audi Group

ZrO: Óxido de Zircônia

1 INTRODUÇÃO

1.1 Motivação

A tecnologia é um fator essencial para a indústria automotiva. O crescente avanço tecnológico e a conversão entre as diferentes tecnologias existentes estão propiciando uma integração cada vez maior entre essas tecnologias e os automóveis. Hoje, todos os sistemas de gerenciamento automotivo são eletrônicos, e os sistemas de conveniência e conforto estão mais amigáveis e cada vez mais parecidos com os dispositivos de entretenimento usados em nosso dia a dia. A tendência no futuro é uma total integração entre o automóvel e as diversas tecnologias existentes**.**

Apesar de toda essa integração tecnológica, há um grande legado de automóveis que não dispõem dela, tendo apenas alguns componentes de gerenciamentos eletrônicos mínimos, que são exigidos em lei, e nenhum equipamento eletrônico que proporcione alguma conveniência ou conforto ao motorista.

Analisando esta situação, é possível criar dispositivos para atender esse nicho, através da integração da tecnologia disponível com esses automóveis, para fornecer serviços de alta tecnologia.

Aproveitando este contexto, o projeto propõe a integração de um automóvel com pouca tecnologia embarcada com a tecnologia atual, para a elaboração de um sistema de monitoramento do sistema de gerenciamento do motor, que disponibiliza ao usuário, informações de bordo e dados da situação do sistema de alimentação do veículo, através de um dispositivo móvel e de uma interface de integração entre o veículo e a aplicação, composta por uma interface de conexão e um notebook.

1.2 Objetivo Geral

O objetivo deste projeto é a elaboração de um sistema computacional auxiliar ao veículo, formado por uma interface de hardware, um software embarcado em um *notebook* e um software em dispositivo móvel, que disponibilize ao motorista diversas informações de bordo relativas ao sistema de alimentação do veículo. As informações obtidas pelo dispositivo proporcionam ao condutor do veículo um melhor gerenciamento do comportamento do veículo, contribuindo para a melhoria da dirigibilidade e um melhor acompanhamento das condições de funcionamento do veículo, reduzindo a necessidade de acompanhamento técnico apenas para a identificação de algum defeito no sistema de gerenciamento do motor do veículo.

O sistema deve receber os dados do sistema de injeção eletrônica do veículo através da sua tomada de diagnóstico. A interface trata o sinal de saída da tomada de diagnóstico para o padrão serial, e transmite os dados a um notebook, o software embarcado no notebook trata os dados e gerencia a comunicação com o veículo, e transmite as informações tratadas ao dispositivo móvel através de uma conexão serial *Bluetooth*, que por sua vez possui uma aplicação que interpreta os dados obtidos, e mostra ao condutor do veículo a informação padronizada para leitura.

O sistema de hardware/software aqui proposto tem escopo acadêmico, ou seja, não se pretende neste projeto desenvolver uma aplicação automotiva comercial. O projeto se aplica a veículos com sistema de gerenciamento eletrônico do motor, que utilize o protocolo *Key Word 1281* e com pouca tecnologia embarcada, no qual se torna útil a disponibilização de informações adicionais.

1.3 Objetivos Específicos

Especificamente, o projeto tem como objetivo disponibilizar informações adicionais sobre o funcionamento do veículo, a qualquer instante, sem que o motorista necessite de auxilio técnico para identificar uma falha no sistema de injeção eletrônica do veículo. Por meio de uma aplicação em dispositivo móvel e uma interface de integração entre o veículo e a aplicação, composta de um software embarcado em um notebook e uma interface de hardware para

integração com o veículo, pretende-se demonstrar através do projeto, a leitura dos estados de funcionamento dos dispositivos do sistema de injeção eletrônica veicular. Espera-se com isto:

- Atuar de forma preventiva na manutenção do veículo;
- Melhoria da condução do veículo através de novas informações disponíveis;
- Economia de combustível, através do monitoramento em tempo real do funcionamento do sistema de alimentação.

1.4 Justificativa e Relevância do Tema

Sistemas de monitoramento veicular são ferramentas que propiciam um melhor controle do comportamento do veículo, através da mensuração de valores em tempo real e sua comparação com os dados esperados, assim é possível aferir o desempenho do automóvel e pode-se indicar algum problema encontrado, e ter um melhor controle sobre a condução do veículo.

Nos automóveis, sistemas de monitoramento são muito importantes para garantir seu funcionamento dentro dos padrões pré-estabelecidos, que hoje são muito rígidos, principalmente no que tange à emissão de poluentes na atmosfera. Portanto, é impossível atender esses padrões sem ter um monitoramento eletrônico em tempo real do funcionamento dos sistemas de um automóvel.

A utilização de um sistema de monitoramento veicular que visa informar ao usuário as informações de bordo e diagnóstico do veículo é importante para informar ao condutor se existe algum problema no funcionamento do veículo, e acionar a manutenção, caso necessário. Assim, a assistência eletro-mecânica torna-se desnecessária para a identificação de qualquer erro que esteja armazenado na memória do sistema de injeção eletrônica do veículo, mas vale ressaltar que a atuação na correção do problema continua sob a responsabilidade das assistências mecânicas. O projeto visa apenas à identificação do erro armazenado na memória da UCE do veículo e sua disponibilização ao usuário, em um veículo que não possui esta funcionalidade.

1.5 Escopo do Trabalho

O projeto visa coletar as informações do sistema de alimentação do veículo através da tomada de diagnóstico veicular padrão OBDII (*On-Board Diagnosis II*), tratar o sinal de saída em um circuito eletrônico (interface), e converter o sinal para o padrão serial USB. Esses dados são transmitidos a um *notebook*, com uma aplicação que captura as informações da tomada de diagnostico, por meio da interface, trata esses dados e envia a informação a um dispositivo móvel, que mostra ao usuário de uma forma mais amigável os dados obtidos. O software embarcado no dispositivo móvel é feito de forma evitar ao máximo a perda de dirigibilidade devido ao manuseio deste equipamento

Assim, o projeto aplica-se aos veículos de passeio que possuem central eletrônica, interface de diagnóstico OBDII, protocolo de comunicação de dados KW1281 (*Key Word* 1281), e que carecem de informações de condução do veículo, só possuem o indicador de velocidade, temperatura do líquido de arrefecimento do motor, indicador de nível de combustível, indicador de pressão do óleo do motor e indicador de problema no sistema de carga do veículo.

1.6 Resultados Esperados

Pretende-se por meio deste projeto implementar em um veículo um sistema de monitoramento veicular, com a aquisição das informações da central eletrônica do veículo, através da tomada de diagnóstico, e a disponibilização dessa informação ao condutor do veículo. Para isso é utilizada uma interface de comunicação que trata o sinal para o padrão serial USB. Um notebook recebe a informação através da porta USB e uma aplicação trata as informações e controla a comunicação. Após esse processo, a informação é transmitida através de conexão *Bluetooth* para o dispositivo móvel que disponibiliza a informação para o usuário. Espera-se com isto auxiliar o motorista na manutenção de seu veículo, sendo alertado durante a execução do software do sistema sobre a ocorrência de algum defeito no sistema de gerenciamento do motor.

1.7 Estrutura do Trabalho

Esta monografia está estruturada em seis capítulos. O primeiro é a Introdução, que trata da motivação deste projeto, os objetivos gerais e específicos, a justificativa e relevância do tema proposto, o escopo do trabalho, os resultados esperados e a estrutura do trabalho.

O segundo capítulo, Apresentação do Problema, apresenta com base no problema identificado a descrição do trabalho, pesquisa e proposta de solução

No terceiro capítulo, Referencial Teórico e Bases Metodológicas, são apresentdas as metodologias técnicas e tecnologias e outras ferramentas utilizadas na formação do projeto proposto. Ele está organizado em temas essenciais que referenciam o projeto final, são eles: motor de combustão interna; sistema de gerenciamento do motor; diagnose *on-board*; *keyword protocol* 1281, padrão ASCII e comunicação *Bluetooth.*

O quarto capítulo, Proposta de Solução e Modelo, apresenta a topologia do projeto e a descrição dos seus elementos. Este capítulo descreve detalhadamente em seqüência, cada etapa da implementação do projeto.

No quinto capítulo, Aplicação da Solução com Resultados, é feita uma análise geral dos resultados obtidos, das dificuldades encontradas e das recomendações a serem sugeridas para aplicações futuras. São apresentados também a avaliação global do modelo de sua aplicabilidade, pontos fortes e fracos, limitações e resultados mais importantes.

No sexto capítulo, é apresentada a conclusão do trabalho como um todo.

2 APRESENTAÇÃO DO PROBLEMA

Existem diversos veículos que não apresentam informações de bordo ao condutor do veículo, e quase nenhum veículo informa ao usuário as informações de diagnóstico de forma completa, que é a indicação de qual o componente defeituoso do sistema de alimentação do veículo. E todos os veículos com sistema de injeção eletrônica de combustível produzidos no Brasil e na maior parte do mundo têm, por lei, que possuir a chamada tomada de diagnóstico, que nada mais é que uma interface de comunicação entre a UCE (*Unidade de Controle Eletrônico*) do Veículo e um dispositivo de diagnóstico externo.

Ferramentas de diagnose são fundamentais durante o desenvolvimento de novos veículos e sistemas eletrônicos, assim como durante a realização dos procedimentos de revisão e manutenção. [GUIMARÃES, 2007]

Segundo publicação da *Verso Comunicação* (2009), o índice de manutenção preventiva do veículo cai de acordo com o aumento da idade do veículo. A Pesquisa revela que em cada 10 veículos com idades entre 10 e 15 anos apenas 4 fazem manutenção preventiva, já os mais novos, com até dois anos, de 10 veículos 7 vão à oficina para receber cuidados preventivos. A soma dos veículos que não fazem manutenção com idades entre 10 e 15 anos totalizam mais de 5 milhões de unidades, volume superior a frota circulante de automóveis dos Estados do Rio de Janeiro, Minas Gerais e Bahia que equivale a 4,9 milhões.

Essa constatação é preocupante, pois com o uso e o aumento da idade, há um desgaste natural das peças que precisam ser repostas por outras de qualidade, conforme recomenda o manual do fabricante para garantir as boas condições do veículo. “O motorista brasileiro ainda não tem o hábito de cuidar preventivamente de seu veículo, muitas vezes, isso acontece por falta de informação sobre o assunto”. [BENTO, 2009]

Com base nas informações citadas acima é de extrema importância que o veículo esteja funcionando de acordo com os padrões estabelecidos pelo fabricante, isso garante que não sejam emitidos poluentes à atmosfera além do estabelecido e a minimização dos acidentes devido a alguma falha no veículo.

Como foi citado, a divulgação das informações de diagnóstico e de bordo ao condutor do veículo é fundamental para que o mesmo tenha consciência do real funcionamento do veículo e realize as manutenções preventivas/corretivas de forma correta.

Nos veículos com idade entre 10 e 15 anos, as informações de bordo e diagnóstico em sua maioria, não são informadas de forma completa. Vêem-se muitos veículos com apenas os indicadores básicos de velocidade, nível de combustível, indicadores de pressão do óleo do motor e indicador de temperatura do líquido de arrefecimento, e que não possui nenhum indicador de diagnostico que informe algum problema no sistema de injeção eletrônica do veículo.

Para adquirir informações adicionais nesses veículos, ou o condutor procura a assistência técnica para obter essas informações pontualmente, ou adquire produtos de informação de bordo adicionais como conta-giros, manômetro, voltímetro. Um desses equipamentos é exemplificado na figura 2.1.

Figura 2.1. Conta Giros veicular. (Fonte: http://www.americanas.com.br/AcomProd/17278/247751)

Para as informações de diagnóstico, são oferecidos no mercado os testers ou scanners, como o apresentado na figura 2.2, que são vendidos para assistências mecânicas e têm o objetivo de identificar falhas no sistema de alimentação do veículo. Estes equipamentos por sua especificidade possuem o preço bastante elevado.

Figura 2.2. Equipamento de Diagnóstico (tester) multimarca. (Fonte: http://www.deltaferramentas.com.br/loja/produtos_descricao.asp?popupAddCarrinho=S&codigo_produto=33)

Para a minimização deste problema o projeto visa a elaboração de um sistema de hardware/software, embarcado ao veículo que utilize as informações da tomada de diagnóstico do veículo para disponibilizar ao usuário as informações de condução do veículo e as informações de diagnóstico.

O sistema proposto se comunica com o veículo através da porta de diagnóstico do veículo. Para isso é necessário o entendimento do protocolo de comunicação kw1281, para poder estabelecer conexão com a UCE do veículo e adquirir as informações de bordo e diagnóstico.

Todo controle da comunicação fica a cargo do software embarcado no notebook, o software tem a função de adquirir as informações do veículo, tratar essas informações e transmitir essas informações ao usuário por meio de uma conexão *Bluetooth*. O dispositivo móvel possui uma aplicação que mostra ao condutor a informação padronizada de forma apropriada, minimizando a perda de dirigibilidade do condutor ao manipular este sistema.

Dessa forma, o condutor tem a sua disposição informações adicionais que proporcionam uma melhor visão do comportamento do veículo, podendo-se assim, conduzir o veículo de uma forma mais racional. Também é possível verificar se existe alguma avaria no sistema de injeção eletrônica do veículo, contribuindo para uma melhor manutenção do veículo.

3 REFERENCIAL TEÓRICO

3.1 Motor de Combustão Interna

O Motor de combustão interna é a fonte de energia usada com mais freqüência em veículos automotores. Os motores de combustão interna geram energia através da conversão de energia química contida no combustível em calor e o calor assim produzido, em trabalho mecânico. A conversão de energia química em calor é realizada através de combustão, enquanto a conversão subsequente em trabalho mecânico é realizada permitindo-se que a energia do calor aumente a pressão dentro de um meio, que então realiza trabalho à medida que se expande. [BOSCH, 2005]

Segundo Stone (1999, tradução nossa), os dois principais tipos de motores de combustão interna são: ignição por centelha (*SI-Spark Ignition*), onde a combustão é iniciada por uma centelha; e ignição por compressão (*CI - Compression Ignition*), onde o aumento da temperatura e pressão durante o processo de compressão é suficiente para causar a combustão espontânea do combustível. O motor de ignição por centelha é conhecido como o motor a gasolina, a álcool ou a gás, que são os combustíveis típicos desse tipo de motor. O motor de ignição por compressão é conhecido como motor a diesel, por causa de seu combustível utilizado.

Segundo Stone (1999, tradução nossa) ambos os tipos de motores tem quatro ciclos de operação, esse processo é conhecido como ciclo Otto, e é o que determina o funcionamento do motor. Conforme a figura 3.1, os quatro ciclos de operação do motor são:

- **Admissão**: Processo onde a válvula de admissão se abre, e o pistão do cilindro desce, enchendo-o com a mistura de ar e combustível;

- **Compressão**: Ambas as válvulas estão fechadas, e o pistão move-se em direção ao topo do cilindro. Como o pistão se aproxima do ponto máximo do cilindro (PMS – Ponto morto superior), a combustão ocorre;

- **Expansão**: A combustão aumenta a temperatura e pressão dentro do cilindro, e força o pistão para baixo. Este é o processo onde o trabalho que movimentará o veículo é gerado.

- **Exaustão ou escape**: A válvula de escape abre, e os gases resultantes da queima da mistura ar combustível são exauridos do interior do cilindro para o meio externo.

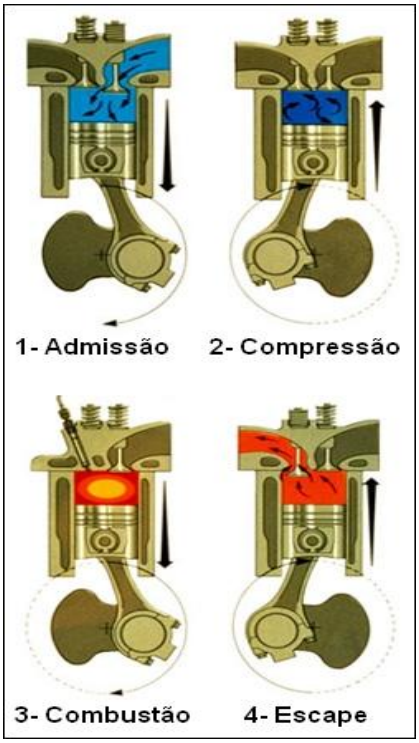

Figura 3.1. As quatro fases do ciclo otto em um motor de combustão interna. (Fonte: www.mecanica.ufrgs.br/mmotor/4tempos.jpg)

3.2 Sistema de Gerenciamento do Motor

3.2.1 Introdução

O gerenciamento do motor se encarrega de converter o desejo do motorista, p. ex., aceleração para uma determinada potência do motor Otto. Ele regula todas as funções do motor de tal maneira que o torque desejado esteja disponível com o consumo e emissões reduzidas. [BOSCH, 2005]

A função principal do gerenciamento de motor é coordenar os diversos sistemas parciais, para ajustar o torque gerado pelo motor e satisfazer ao mesmo tempo as altas exigências quanto a: [BOSCH, 2005]

- Emissão de gases de escape;
- Consumo de combustível;
- Potência;
- Conforto;
- Segurança;

3.2.2 Sensores

3.2.2.1 Sensor de Concentração de Oxigênio (Sensor Lambda de Oxigênio)

O sistema medidor de combustível emprega o conteúdo residual do oxigênio no escapamento, conforme medido pelo sensor lambda de oxigênio para regular bem precisamente a mistura ar/combustível para combustão, ao valor lambda $(\lambda) = 1$. [BOSCH, 2005]

O sensor é um eletrólito em estado sólido, feito de material cerâmico ZrO (Óxido de Zircônia). Sob altas temperaturas, esse eletrólito torna-se condutor e gera uma carga galvânica característica nas conexões do sensor. Essa tensão é um índice do conteúdo do oxigênio no gás. A variação máxima ocorre em λ=1. [BOSCH 2005].

Para o tratamento catalítico do gás de escape por um catalisador de três vias é imprescindível a manutenção exata de λ=1 com o motor aquecido. Para conseguir isso, é necessário determinar exatamente a massa de ar admitido e adicionar uma quantidade de combustível exatamente dosada. Pode-se observar na figura 3.2 o típico fucionamento do sensor de oxigênio. [BOSCH 2005]

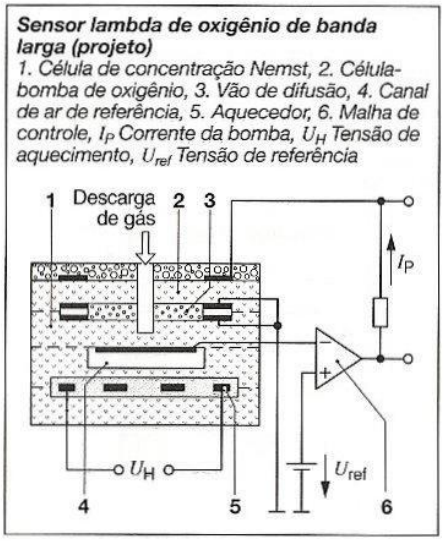

Figura 3.2. Sensor de Oxigênio. (Fonte: BOSCH, 2005)

3.2.2.2 Medidor de Massa de Ar

Medidores de massa de ar funcionam de acordo com o princípio de filme ou fio aquecido; a unidade não contém partes mecânicas móveis. Conforme a figura 3.3, o circuito de controle em malha fechada no alojamento do medidor mantém um diferencial constante de temperatura entre um fino fio de platina ou resistor de filme fino e a corrente de ar que passa. A corrente exigida para o aquecimento fornece um índice extremamente preciso, embora não linear de taxa de fluxo de massa de ar. Geralmente o sistema UCE converte os sinais na forma linear e assume outras tarefas de avaliação de sinal. [BOSCH, 2005]

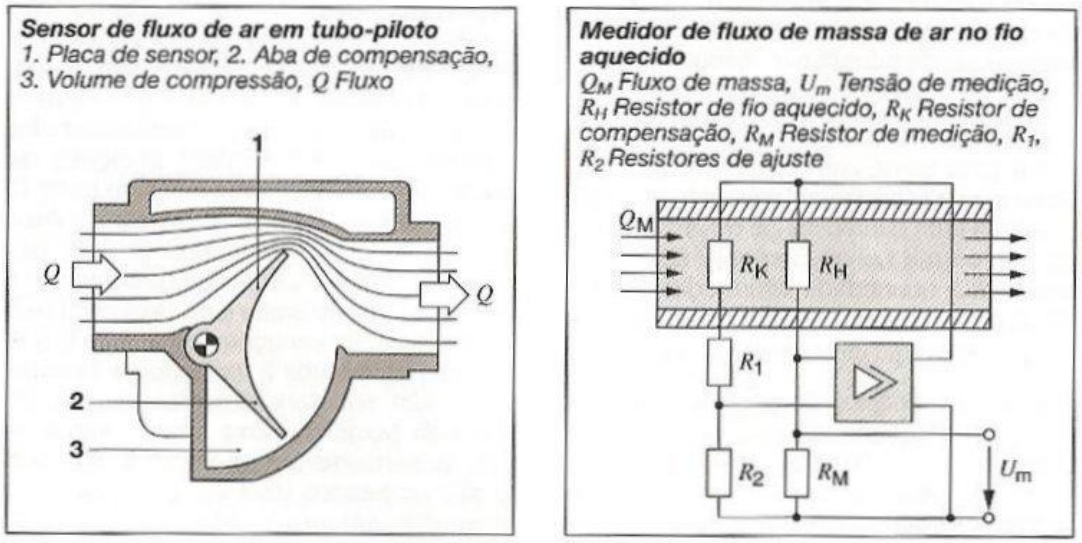

Figura 3.3. Medidores de massa de ar. (Fonte: BOSCH, 2005)

3.2.2.3 Sensor de Temperatura

Medições de temperatura em veículos a motor são realizadas pela exploração da sensibilidade à variação de temperatura, encontrada na resistência elétrica dos materiais com coeficiente de temperatura positivo (CTP) ou negativo (CTN), como termômetros de contato. Conforme a figura 3.4, a conversão da variação de resistência em tensão analógica é predominantemente desempenhada com a ajuda de resistores neutros de temperatura complementares ou inversamente sensíveis, como divisores de tensão (também fornecendo linearidade aumentada). [BOSCH, 2005]

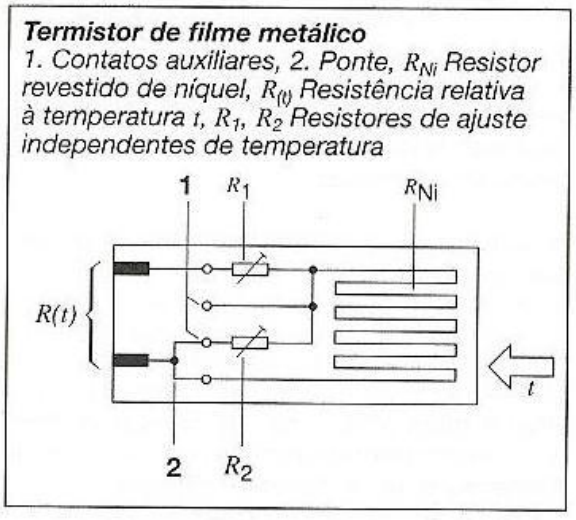

Figura 3.4. Sensor de Temperatura. (Fonte: BOSCH, 2005)

3.2.2.4 Sensores de Velocidade e RPM

Medições são basicamente tomadas com a ajuda de um sistema incrementado de sensor, e consistem de engrenagem e sensor de velocidade rotativa. [BOSCH, 2005].

O sensor indutivo consiste de um ímã de barra, com um pino ferromagnético, sustentando uma bobina de indução com dois terminais, como pode ser visto na figura 3.5. Quando uma engrenagem de anel ferromagnético (ou rotor de projeção semelhante) passa por este sensor, gera uma tensão na bobina, tensão esta diretamente proporcional à variação periódica no fluxo magnético. Um padrão uniforme de dentes gera uma curva de tensão quase senoidal. A velocidade rotativa reflete-se no intervalo periódico entre os pontos de transição por zero da tensão, ao passo que a amplitude também é proporcional à velocidade rotativa. [BOSCH, 2005]

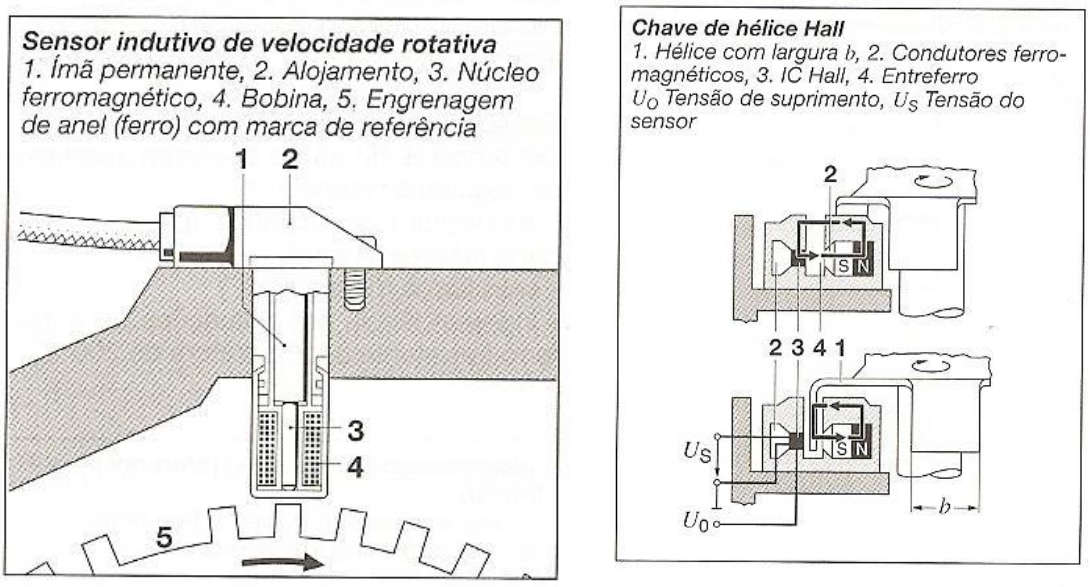

Figura 3.5. Sensor do tipo indutivo (esquerda) e sensor hall (direita). (Fonte: BOSCH, 2005)

3.2.3 Atuadores

Atuadores (elementos finais de controle) formam uma interface entre o processador de sinal eletrônico (processamento de dados) e o processo real (movimento mecânico). Convertem sinais de baixa potência que transmitem informação sobre posicionamento em sinais operacionais de um nível de energia adequado ao controle do processo. Os conversores de sinais são combinados com elementos amplificadores para explorar os princípios de transformação física que regulam as inter-relações entre as várias formas de energia (elétrica-mecânica-fluída-térmica). [BOSCH, 2005]

3.2.3.1 Válvula Injetora para Injeção no Coletor de Admissão

As válvulas injetoras consistem principalmente de: [BOSCH, 2005]

- Uma carcaça da válvula com bobina magnética e conexão elétrica;
- Um assento de válvula com placa de furos;
- Uma válvula de agulha móvel com induzido magnético.

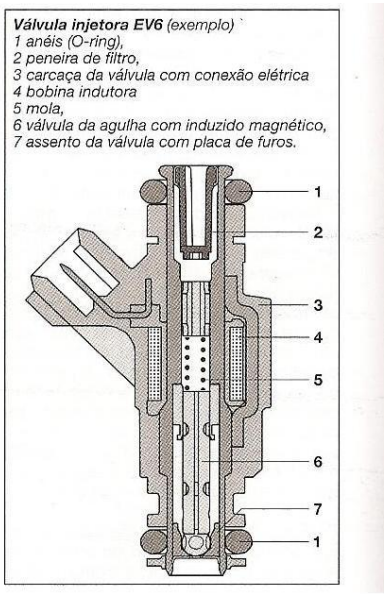

Figura 3.6. Válvula injetora de combustível. (Fonte: BOSCH, 2005)

Conforme a figura 3.6, uma peneira de filtro na entrada do combustível protege a válvula injetora de impurezas. Dois anéis (O-ring) vedam a válvula contra a galeria de combustível e o coletor de admissão. No caso de bobina sem corrente, as molas e a força resultante da pressão do combustível pressionam a agulha da válvula sobre o assento da válvula e vedam o sistema de alimentação de combustível contra o coletor de admissão. [BOSCH, 2005]

Quando a válvula injetora é alimentada, a bobina produz um campo magnético. O induzido é atraído pelo campo magnético, a agulha da válvula se levanta do assento e o combustível flui através da válvula injetora. [BOSCH, 2005]

O volume de combustível injetado por unidade de tempo é determinado principalmente pela pressão do sistema e do diâmetro livre dos furos de injeção na placa. Quando a corrente de excitação é desativada, a agulha da válvula fecha novamente. [BOSCH, 2005]

3.2.3.2 Borboleta de Aceleração

O elemento atuador central para a influência do fluxo de massa de ar é a borboleta de aceleração. Quando a borboleta de aceleração não está totalmente aberta, o ar aspirado pelo motor é estrangulado e o torque produzido é reduzido. Esse efeito estrangulador depende da posição e, portanto, do diâmetro da abertura da borboleta de aceleração, bem como da rotação do motor. Com a borboleta de aceleração totalmente aberta é atingido o torque máximo do motor. [BOSCH, 2005]

Conforme a figura 3.7, em sistemas com acelerador eletrônico EGAS é determinado, a partir do torque desejado do motor (posição do pedal do acelerador), o enchimento de ar necessário para isso e então aberta a borboleta de aceleração. [BOSCH, 2005]

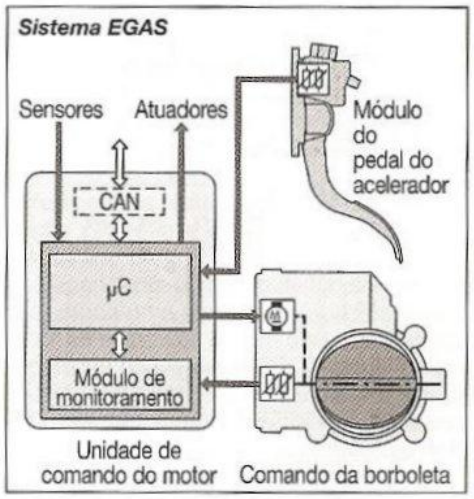

Figura 3.7. Sistema EGAS

Em sistemas convencionais, o motorista comanda diretamente a abertura da borboleta de aceleração através do acionamento do pedal do acelerador. [BOSCH, 2005]

3.2.3.3 Bomba elétrica de combustível

A bomba elétrica de combustível deve disponibilizar ao motor, sob todas as condições operacionais, a quantidade suficiente de combustível com a pressão necessária para a injeção. Os componentes da bomba de combustível são apresentados na figura 3.8. [BOSCH, 2005]

As principais exigências são: [BOSCH, 2005]

- Vazão entre 60 e 250 l/h com tensão normal;
- Pressão no sistema de combustível entre 300 e 650 KPa
- Aumento da pressão a partir de 50..60% da tensão nominal: determinante para isso é o funcionamento na partida a frio.

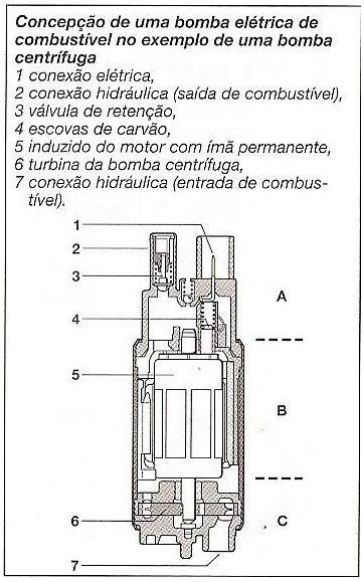

Figura 3.8. Bomba de Combustível. (Fonte: BOSCH, 2005)

3.2.4 Módulos Eletrônicos

Os módulos eletrônicos são os dispositivos responsáveis pela leitura das entradas, acionamento das saídas e pelo gerenciamento do funcionamento dos protocolos de comunicação utilizados nos veículos. [GUIMARÃES, 2007]

Um módulo eletrônico automotivo é como um computador. Possui internamente uma placa de circuito impresso com um microprocessador ou microcontrolador e um programa gravado em uma memória, isso pode ser observado na figura 3.9. Em função do estado das entradas conectadas ao módulo, o software decide o que realizar com as saídas. [GUIMARÃES, 2007]

Existem vários tipos de módulos de controle, basicamente diferenciados pelas funções realizadas e pelas suas características técnicas, especialmente em relação ao software. [GUIMARÃES, 2007]

3.2.4.1 Conceituação Técnica

uP/uC = Microcontrolador ou microprocessador: é o cérebro do módulo eletrônico, responsável pela execução dos programas e pelo processamento e controle das atividades do módulo.

Mem = memória: armazena o programa do módulo eletrônico. No geral, é o tipo PROM ou Flash. O primeiro caso é caracterizado por programa já gravado do fornecedor e não pode mais ser alterado (processo conhecido como mascaramento). O segundo caso é caracterizado por permitir a troca do programa do módulo a qualquer momento, procedimento comumente utilizado em algumas aplicações ou situações especiais, que necessitam da atualização da versão do programa. [GUIMARÃES, 2007]

Entradas: Porta de entrada dos sinais digitais e/ou analógicos medidos pelos transdutores (sensores). [GUIMARÃES, 2007]

Saídas: porta de saída dos sinais digitais e/ou analógicos controlados pelo módulo eletrônico. [GUIMARÃES, 2007]

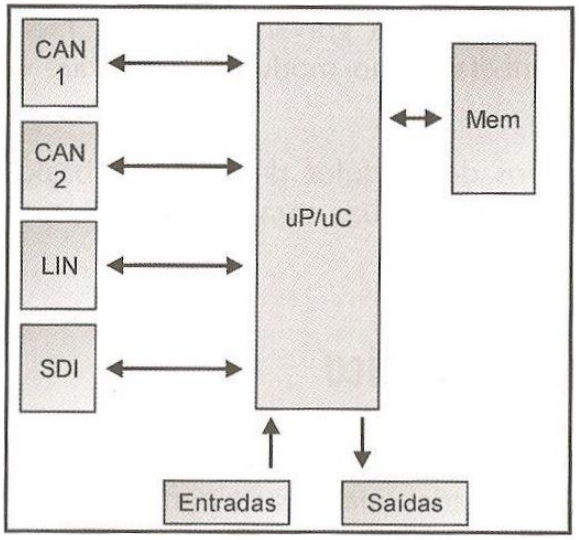

Figura 3.9. Diagrama de bloco de uma UCE. (Fonte: GUIMARÃES, 2007)

3.2.5 Gerenciamento de Motor Motronic

Motronic é o nome dado pela Bosch para sistemas de controle e regulagem do motor Otto. Originalmente a Motronic tinha a função básica de combinar a injeção eletrônica com uma ignição eletrônica em uma unidade de comando. Gradualmente foram adicionadas mais funções que se fizeram necessárias em função das exigências das leis para redução de emissão de gás de escape, redução do consumo de combustível, aumento da demanda de potência, conforto e segurança ao dirigir. [BOSCH, 2005]

Uma outra função importante da Motronic é a monitoração da funcionalidade do sistema geral de diagnóstico "on board" (OBD). Através das exigências legais são impostas exigências a Motronic, que resulta em torno da metade da capacidade do sistema Motronic (em termos de poder de computação e necessidade de memória) seja dedicada ao diagnóstico. [BOSCH 2005]

O sistema Motronic contém todos os sensores para detectar os dados operacionais do motor e do veículo, bem como todos os atuadores para executar as intervenções de ajuste no motor Otto. A unidade de comando usa os dados dos sensores para captar o estado do motor e do veículo em intervalos muito curtos (a faixa de milissegundos para atender as exigências em tempo real do sistema). Circuitos de entrada suprimem as interferências dos sinais dos sensores e os colocam em uma faixa de tensão uniforme. Um conversor analógico-digital transforma então os sinais preparados em valores digitais. Outros sinais são captados através das interfaces digitais (p. ex. Bus CAN) ou interfaces moduladas por largura de pulso (PWM – Pulse Width Modulation). [BOSCH, 2005]

A parte central da unidade de comando do motor é um microcontrolador com a memória de programa (EPROM ou flash), na qual estão armazenados todos os algoritmos de comando de operações – são cálculos baseados num determinado esquema – e dados (parâmetros, curvas características, mapas característicos). As grandezas de entrada derivadas dos sinais dos sensores influenciam os cálculos nos algoritmos e com isso os sinais de comando para os atuadores. O microcontrolador reconhece baseados nesses sinais de entrada, a reação do veículo desejada pelo motorista e calcula a partir disso: [BOSCH, 2005]

- O torque necessário;

- O enchimento dos cilindros resultantes e a respectiva quantidade injetada;
- A ignição no tempo certo;
- Os sinais de comando para os atuadores, p. ex. do sistema de retenção de evaporação de combustível, do turbo compressor do gás de escape, do sistema de ar secundário

Os dados do sinal de baixo nível disponíveis nas saídas do microcontrolador são adaptados pelos estágios de saída de potência aos níveis necessários para os respectivos atuadores [BOSCH, 2005].

A M-Motronic abrange todos os componentes necessários para o controle de um motor com injeção no coletor de admissão e borboleta de aceleração convencional [BOSCH, 2005].

O motorista ajusta o fluxo de massa de ar e com isso o torque diretamente através do pedal do acelerador e da borboleta aceleradora. A Motronic adapta a necessidade de ar, p. ex. com baixa temperatura do motor ou para regulagem de marcha lenta usando o atuador de ar bypass. A partir da corrente de massa de ar aspirada, captada com a ajuda de sensores de carga (p. ex. medidor de massa de ar, sensor de pressão do coletor de admissão), a M-Motronic calcula a massa de combustível necessária, bem como o melhor ponto de ignição possível para o ponto de funcionamento ajustado. Um funcionamento otimizado do motor consegue-se com a ativação das válvulas de injeção e bobinas de ignição no tempo certo [BOSCH 2005].

3.3 Diagnose On-Board (On Board Diagnostic – OBD)

O termo diagnose veicular (ou diagnóstico veicular) representa as funções ou ferramentas que permitem a programação ou verificação do funcionamento de cada módulo eletrônico existente em um veículo. Com o aumento da eletrônica embarcada, passa a ser mandatório o desenvolvimento de dispositivos que, por exemplo, permitam o diagnóstico de falhas eventuais dos sistemas. [GUIMARÃES, 2007]

Considerando esta necessidade, podemos classificar as falhas em duas categorias: as possíveis de serem identificadas pelo motorista e as identificadas somente com o auxílio de ferramentas especiais. A primeira é chamada de *On-Board Diagnosis* (OBD) e é realizada por

meio de avisos sonoros e lâmpadas específicas existentes no painel de instrumentos. A segunda pode ser chamada de *Off-Board Diagnosis* e é realizada pelos chamados *Testers*, dispositivos eletrônicos capazes de se comunicar com os módulos do veículo sistemas. [GUIMARÃES, 2007]

Segundo KIENCKE e NIELSEN (2005, tradução nossa), diagnósticos de motores automotivos têm uma longa história. Desde o primeiro motor automotivo do século XIV, era necessário encontrar defeitos nos motores. Por um longo tempo, os diagnósticos eram realizados manualmente, mas ferramentas de diagnóstico começaram a aparecer na metade do século XX. Um exemplo é o estroboscópio que é usado para determinar o tempo de ignição. Na década de 60, a mensuração do sistema de escape se tornou uma forma comum de se diagnosticar o sistema de combustível. Até 1980 todos diagnósticos eram feitos manualmente e de forma off-board. Foi nessa época que a eletrônica e os microprocessadores gradualmente foram introduzidos nos carros. Essa abertura possibilitou o uso do diagnóstico *on-board*. O objetivo foi criar uma forma fácil de encontrar falhas.

Pode-se definir a diagnose *on-board*, basicamente, como a leitura das falhas dos sistemas do veículo por meio do painel de instrumentos. [GUIMARÃES, 2007]

Entre os diversos objetivos procurados pela diagnose *on-board*, destacam-se aqueles ligados diretamente ao motorista ou ao condutor do veículo, como por exemplo: [GUIMARÃES, 2007]

- Alertar o motorista sobre o funcionamento inadequado dos sistemas, mesmo não havendo um sintoma de anomalia perceptível;
- Facilitar ao motorista a identificação de falhas em caso de pane do veículo;
- Auxiliar o motorista quanto à correta utilização dos diversos sistemas (mecânicos e eletroeletrônicos) do veículo;

Dentre os principais desafios que se apresentam para que um sistema de diagnose *on-board* possa ser considerado eficaz, destacam-se: [GUIMARÃES, 2007]

- Fácil acesso às informações;

- Fácil interpretação das informações, não deixando dúvidas;
- Não transmitir alerta maior (ou menor) que o necessário.

Logicamente, um sistema de diagnose *on-board* que vise o desempenho ótimo deve ser projetado em função do conhecimento e até mesmo da cultura de quem opera o veículo. Portanto, fornecer informações insuficientes ou em demasia é também uma questão de ponto de vista, intrinsecamente ligada à formação (técnica e cultural) do condutor do veículo. No caso de veículos comerciais que operam em frotas, podem-se minimizar esses efeitos por meio de treinamento dos operadores e motoristas. [GUIMARÃES, 2007]

É importante adicionar que, quando analisamos a evolução da eletrônica embarcada nos veículos e a evolução dos sistemas de diagnose *on-board*, notamos que ambas estão intimamente relacionadas. O conceito de diagnose *on-board* foi fundamental para o avanço tecnológico que vemos atualmente nos veículos. [GUIMARÃES, 2007]

O componente que mais contribui para a modalidade de diagnóstico embarcado é o painel de instrumentos (*cluster*). [GUIMARÃES, 2007]

Do final de 1920 até o final de 1950, a instrumentação básica dos veículos era composta por velocímetro, indicador de pressão do óleo, temperatura do líquido de arrefecimento do motor e do indicador do nível de combustível. De fato, em muitos veículos, nenhum desses instrumentos possuía, necessariamente, características elétricas, e alguns deles foram simplesmente substituídos por lâmpadas de alarme, em veículos de menor preço e de produção em grande escala. [GUIMARÃES, 2007]

A partir do ano de 1970 teve início a instrumentação eletrônica propriamente dita, ampliando sua função dentro do sistema veicular como um todo. Dentre as funções do painel de instrumentos adicionadas, podemos citar: [GUIMARÃES, 2007]

- Indicador de informações e alertas;
- Gerenciador dos diversos sistemas eletrônicos;
- Provedor de informações de condução ótima do veículo;

- Apresentador de dados relativos à diagnose dos diversos sistemas eletrônicos veiculares;

- Aquisitor de dados de condução, para melhor gerenciamento de frota, entre outras.

Todas essas funções estão relacionadas à diagnose *on-board*, deixando clara a importância do painel de instrumentos na estratégia de diagnóstico.

Segundo KIENCKE e NIELSEN (2005, tradução nossa), em 1988, a primeira regulamentação legal em relação ao diagnóstico on-board, OBD, foi introduzida pelo *CARB – California Air Resource Board*, inicialmente essa regulamentação foi aplicada somente na Califórnia, mas o *Federal Environmental Protection Agency (EPA*) adotou uma regulamentação similar e a aplicou em todo o Estados Unidos. Isso forçou a indústria a introduzir mais e mais capacidade nos diagnósticos on-board dos veículos. Em 1994, uma nova e mais rigorosa regulamentação, a OBDII foi aplicada na Califórnia. Hoje o padrão OBDII está na maioria dos sistemas de gerenciamento do motor. Seguindo o exemplo da Califórnia e dos Estados Unidos, esse tipo de legislação tem sido aplicada em diversos outros países. O conector padrão pode ser observado na figura 3.10.

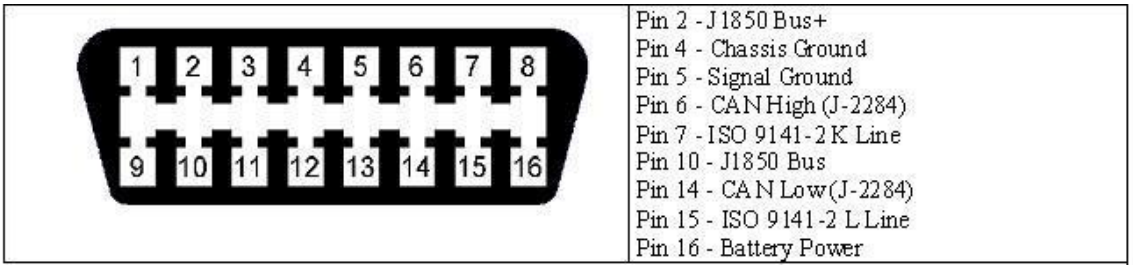

Figura 3.10 Conector padrão OBD II (com adaptações). (Fonte: http://www.obdii.com/connector.html#dates)

De acordo com o parágrafo 1º, do artigo 13º da lei n° 8.723, de 28 de outubro de 1993, "Os fabricantes de veículos automotores ficam obrigados a divulgar às concessionárias e distribuidores as especificações e informações técnicas necessárias ao diagnóstico e regulagem do motor, seus componentes principais e sistemas de controle de emissão de poluentes".

Considerando que a Resolução CONAMA (Conselho Nacional do Meio Ambiente) nº 315, de 2002, estabelece a utilização de Sistemas de Diagnose de Bordo OBD por

constituírem tecnologia de ação comprovada na identificação de mau funcionamento dos sistemas de controle de emissão possibilitando a antecipação de medidas corretivas e a consequente prevenção no aumento da emissão de poluentes atmosféricos. [RESOLUÇÃO CONAMA nº 354, de 13 de dezembro de 2004]

Considerando que a adoção do OBD nos veículos automotores representa expressivo avanço tecnológico que possibilita ao usuário do veículo prevenir a ocorrência de danos severos aos sistemas de controle de emissão, contribuindo para a melhoria da qualidade ambiental, e dessa forma salvaguardar os interesses do consumidor e da sociedade em geral. [RESOLUÇÃO CONAMA nº 354, de 13 de dezembro de 2004]

3.4 Key Word Protocol 1281

Segundo a SAE – *Society of Automotive Engineers* (2008, tradução nossa), o *Key Word Protocol 1281* é um protocolo de comunicação de diagnóstico proprietário do grupo Volkswagen. O protocolo kw1281 necessita uma linha de comunicação bidirecional com o dispositivo externo, essa linha de comunicação é conhecida como *K Line.* Este canal de comunicação pode ser utilizado por diversas Unidades de controle do veículo.

3.4.1 Inicialização

Segundo a SAE, a UCE kw1281 é selecionada de todas as UCE's conectadas a *k line* por uma comunicação da ferramenta de diagnóstico (*Scan Tool)*. Essa comunicação dá-se com o envio pela ferramenta de diagnóstico do endereço da UCE kw1281.

Segundo a SAE, com o intuito de acessar o modo de diagnóstico, uma palavra com o endereço da UCE deve ser transmitida por meio da *K line.* A palavra com o endereço usado para simulação deve ser transmitida no formato de 10 bits, sob uma taxa de transferência de 5 baud, com variação de até de 1%.

Segundo a SAE, a palavra com o endereço deverá ser composta, como descreve a figura 3.11, e na mesma sequência:

1 start bit	(logical "0", low potential, reaction of ECU to 1-0 transition)
7 data bits	(ECU-specific address), starting with the LSB
1 parity bit	(parity odd)
1 stop bit	(logical "1", high potential)

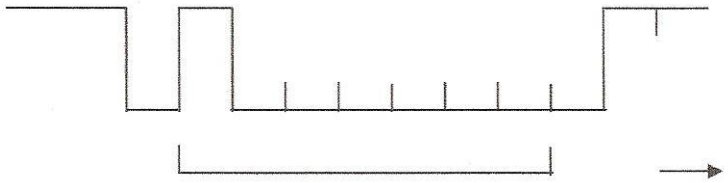

FIGURE 5 - ADDRESS WORD

Figura 3.11. Inicialização da UCE. (Fonte: SAE, 2008)

3.4.2 Byte de Sincronização

Segundo a SAE, a comunicação deste protocolo é baseada nas especificações ISO 9141 e ISO 14230-2.

Segundo a SAE, depois que a UCE recebeu o bit de *stop* na palavra com o endereço identificado como sendo o próprio endereço da UCE, é aguardado pelo período de 1 bit para que a ferramenta de diagnóstico possa se preparar para a comunicação subsequente.

Segundo a SAE, depois de aguardar o tempo, a UCE reage com o envio do byte de sincronização pela *k line*. O byte de sincronização segue a seguinte estrutura, apresentada na figura 3.12:

1 start bit	(logical "0", low potential)
8 data bits	(content = 0x55, starting with the LSB)
1 stop bit	(logical "1", high potential)

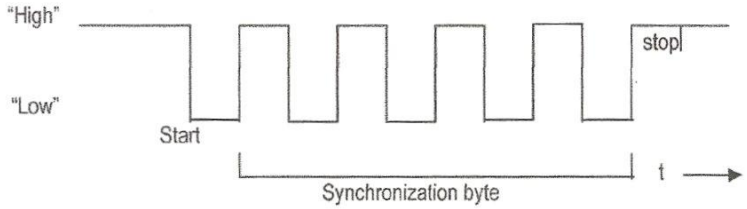

Figura 3.12. Byte de sincronização. (Fonte: SAE, 2008)

Segundo a SAE, o byte de sincronização é transmitido com a mesma taxa de sinalização que é usada pela transferência subseqüente. A taxa de sinalização não deve ser diferente de 10400 baud, com variação de 1%.

3.4.3 Keyword

Segundo a SAE, dois bytes chave são transmitidos depois do byte de sincronização. As palavras têm a seguinte estrutura:

Tabela 3.1. Bytes-chave. (Fonte: SAE)

1 start bit	(logical "0", low potencial, reaction of UCE to 1-0 transition)
7 key data bits	
1 parity bit	(parity odd)
1 stop bit	(logical "1", high potencial)

Estes bytes-chave devem ser considerados como pertencentes a uma informação de 14 bits, nessa informação o byte menos significativo 0x01 é transmitido primeiro, seguido pelo byte mais significativo 0x8A como apresentado na figura 3.13.

Keyword		
	low	high
7 Key data bits	0x01 (000 0001B)	0x0A (000 1010B)
with parity	0x01 (0000 0001B)	0x8A (1000 1010B)

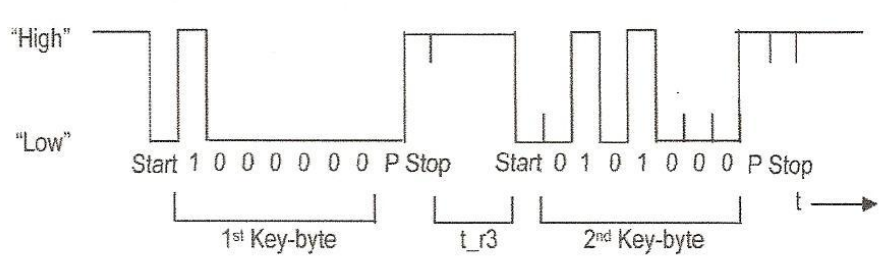

Figura 3.13. Bytes chave. (Fonte: SAE, 2008)

Segundo a SAE, depois da transmissão do segundo byte-chave, a UCE aguarda o envio do complemento do segundo byte-chave pela ferramenta de diagnóstico como confirmação da efetivação da transmissão. Nessa transmissão a ferramenta de diagnóstico deve observar o intervalo de 25 ms antes de enviar o complemento. Este intervalo tem a finalidade de permitir que a UCE tenha tempo suficiente para mudar para o modo de recebimento.

Segundo a SAE, concluído este processo de comunicação, a UCE transmite o primeiro byte do serviço de identificação da UCE.

3.4.4 Formato do Byte de Dados

Segundo a SAE, durante a comunicação, a ferramenta de diagnóstico e a UCE deve operar com o seguinte formato de byte de dados, de acordo com a figura 3.14:

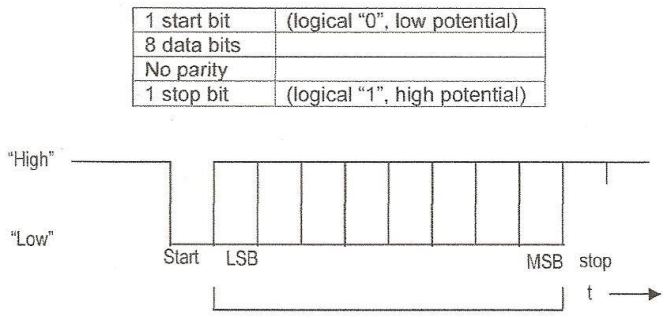

Figura 3.14. Formato do dado. (Fonte: SAE, 2008)

3.4.5 Estrutura de Serviço

Segundo a SAE, as informações que são trocadas entre a ferramenta de diagnóstico e a UCE estão organizada em serviços. Um serviço deve ser combinado de várias mensagens. Todas as mensagens devem ter a mesma estrutura. Cada mensagem consiste de um byte, no qual o tamanho da mensagem é transmitido, um contador da mensagem, a identificação do serviço (*SID – Service Identification*), mensagem específica e o fim da mensagem. Na figura 3.2 é apresentada a estrutura do serviço.

Byte	Content
1	Message length (n-1)
2	Message counter
3	Service Identification
4 ~ n-1	Message-specific information (if necessary)
n	Message end

Tabela 3.2. Estrutura do serviço. (Fonte: SAE, 2008)

3.4.5.1 Tamanho da Mensagem

Segundo a SAE, o byte do tamanho da mensagem especifica o número de bytes subseqüentes que compõe a mensagem.

3.4.5.2 Contador da Mensagem

Segundo a SAE, a primeira mensagem transmitida (é a primeira mensagem de identificação da UCE) tem o contador de mensagem com o valor 1. O valor do contador de mensagem é incrementado a cada mensagem transmitida, este valor varia de 0 a 255. Duas mensagens transmitidas em subseqüência deverão sempre ter direções diferentes de transmissão. O contador de mensagem deverá sempre ser um valor par quando for enviado pela ferramenta de diagnóstico, e um valor par, quando enviado pela UCE.

3.4.5.3 Identificação do Serviço

Segundo a SAE, a identificação do serviço é o terceiro byte a ser transmitido. Ele fornece a informação do serviço dos quais estão sendo enviados a informação ou serviço requisitado.

3.4.5.4 Bytes Específicos da Mensagem

Segundo a SAE, bytes específicos da mensagem são necessários se outra informação é solicitada em adição à identificação do UCE.

3.4.5.5 Byte do Fim da Mensagem

Segundo a SAE, o último byte da mensagem é o byte do fim da mensagem. O byte sempre tem algum conteúdo, normalmente o caractere ASCII "ETX" (*End of Text*), na qual é transmitida com o MSB (*Most Significant Bit*) =0, portanto, representa o número 0x03. A razão para isto é que somente o último byte não é salvaguardado por um handshake. O receptor deverá checar o último byte para estabelecer se o mesmo contém o valor "ETX" com base no tamanho da mensagem recebida. Abaixo, a figura 3.15 exemplifica o fluxo de execução de uma função.

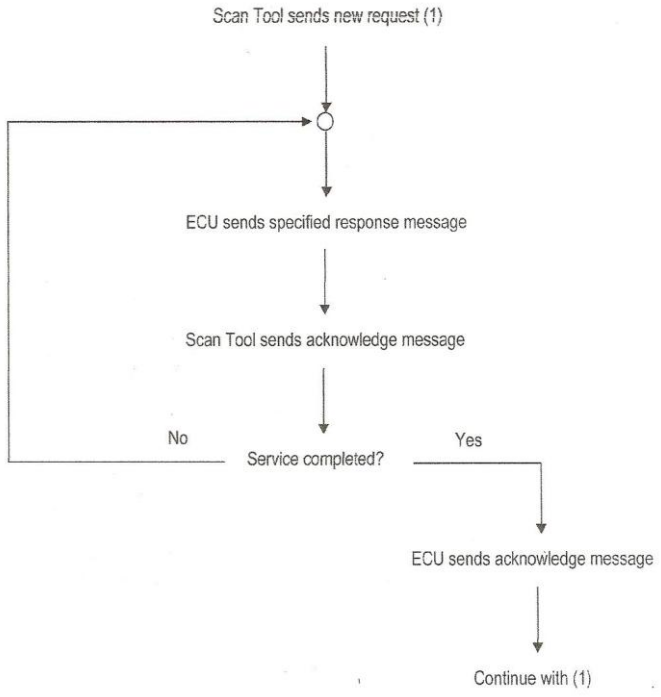

Figura 3.15. Fluxo de execução de uma função. (Fonte: SAE, 20008)

3.5 Padrão ASCII

O *American Standart code for Information Interchange* ou ASCII atribui valores de 0 a 255 para letras maiúsculas, minúsculas, dígitos numéricos, marcas de pontuação e outros símbolos. Caracteres ASCII podem ser divididos dentro das seguintes seções: [COZENS e WAINWRIGHT, 2000]

- 0 – 31: Códigos de controle;
- 32 – 127: Padrão, implementação de caracteres independentes;
- 128 – 255: Símbolos especiais, conjunto de caracteres internacionais

3.6 Bluetooth

Bluetooth é uma tecnologia destinada a servir como padrão universal de conexão entre equipamentos, ou entre equipamentos e seus periféricos por meio de uma faixa limitada de rádio. Os periféricos interconectados podem ser desde terminais móveis, impressoras, organizações pessoais, *notebooks*, computadores pessoais, bem como qualquer tipo de equipamento eletrônico, como máquinas de café, relógios eletrônicos, terminais de pedágio e caixas automáticos, brinquedos de crianças, sistemas de reprodução de som domestico de alta fidelidade, entre outros. Alguns destes periféricos podem ser vistos na figura 3.16 e 3.17. [BERNAL, 2002]

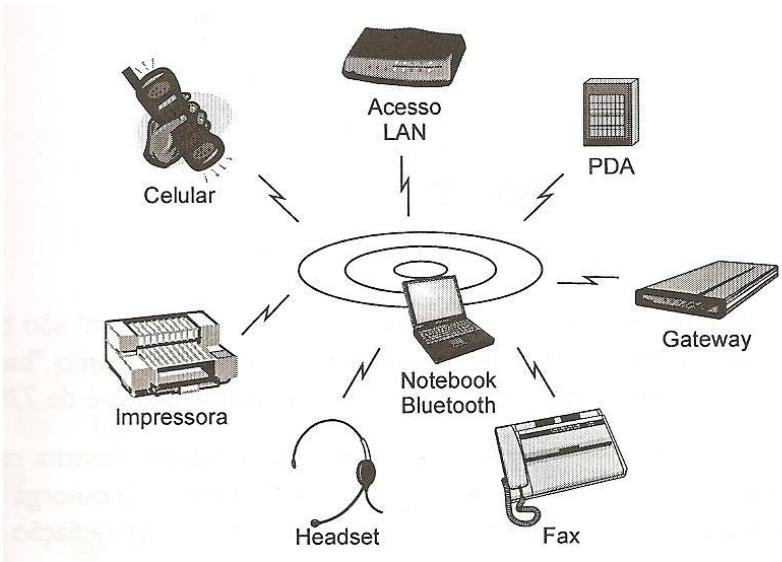

Figura 3.16. Rede local Bluetooth. (Fonte: BERNAL, 2002)

Foi escolhido o nome *Bluetooth* para esse padrão em homenagem ao rei medieval dinamarquês Harald Blaatand II ou *Bluetooth* (940-981). Esse rei se tornou notável por sua luta para unificar a Dinamarca e a Suécia, da mesma forma como os engenheiros modernos

passaram a lutar pela padronização internacional do *Bluetooth*, inicialmente foram cinco companhias promotoras do projeto: Ericsson, IBM, Intel, Nokia e Toshiba. Posteriormente, juntou-se ao grupo a 3Com, Lucent, Microsoft e Motorola. Mais tarde outras 2.164 empresas e entidades passaram a integrar o grupo consorciado. [BERNAL, 2002]

Um trabalho conjunto com o IEEE (Instituto de Engenheiros Eletricistas e Eletrônicos) 802 formou o grupo 802.15.

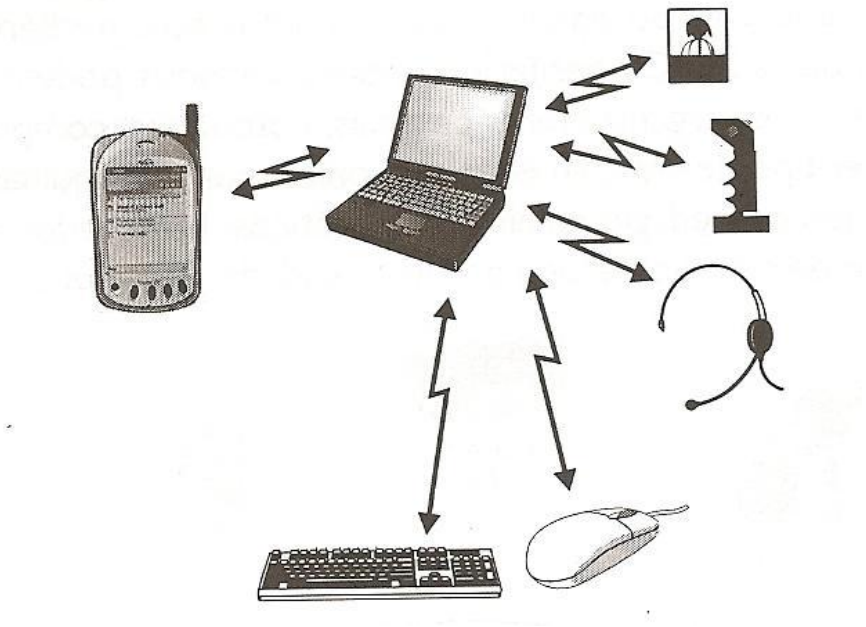

Figura 3.17. Periféricos Bluetooth. (Fonte: BERNAL, 2002)

O sistema *Bluetooth* baseia-se em processos de comunicação, controlados por protocolos, que são transportados pelas interfaces aéreas. As áreas de cobertura dos dispositivos Bluetooth são separadas em conjuntos de até oito dispositivos próximos que compõem uma pico net. Um dos dispositivos do conjunto pico net é definido como "mestre" e provê sinal de relógio como referência e a seqüência de saltos "hop" de dados para sincronizar as comunicações entre todos os membros da pico net. [BERNAL, 2002]

As aplicações potenciais para as redes *Bluetooth* são inúmeras. Aparelhos de som podem ser integrados via canal *Bluetooth* com o fone de ouvido do usuário, aparelhos de medição sendo calibrados automaticamente pelas informações providas pelos servidores, informações sobre pacientes internados em hospitais podem ser transmitidas

ininterruptamente para médicos e enfermeiros, portando em seus cintos terminais móveis, bem como garçons podem anotar pedidos que serão transmitidos diretamente para os servidores de controle de produção, suprimentos e cobrança dos restaurantes mais automatizados. [BERNAL, 2002]

3.6.1 O Uso de Sistemas Bluetooth em Embarcações Automotivas

- Estender as funcionalidades do carro;
- Proporcionar uma interface "homem-máquina", permitindo aos usuários operarem dentro de carros terminais celulares, rádios, fones de ouvido e outros dispositivos de forma "hands free". Permitir o uso de cartões inteligentes SIM do tipo usado nos terminais GSM;
- Os dispositivos portáteis podem exportar sua interface com o usuário para o carro;
- Personalizar a interação e as reações dos dispositivos do veículo, tais como: rádio e celulares. O próprio controle remoto de abertura das portas e desligamento do sistema de alarmes pode ser controlado por meio de comunicação *Bluetooth* e códigos de segurança;
- Acesso a posicionamento de veículos por meio de localização por sensoriamento *Bluetooth;*
- Possibilidade de interação entre o terminal celular e os dispositivos internos do veículo;
- Diagnóstico e programação de dispositivos internos por meio de interação com gigas de teste, diagnóstico e programação, localizados em oficinas, centros automotivos e pontos de acesso fixados pelos fabricantes;
- Os carros embutirão dispositivos de integração com os dispositivos portados internamente pelos usuários do automóvel, bem como trocarão informações com dispositivos externos, localizados pelas vias nas quais transitarão os veículos.

Conforme a figura3.18, é exemplificado um exemplo de aplicação automotiva que utiliza a o padrão de comunicação bluetooth

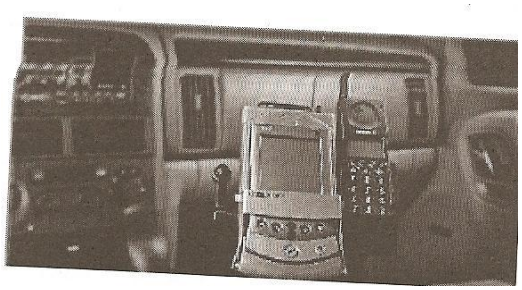

Figura 3.18. Aplicações Bluetooth automobilísticas. (Fonte: BERNAL, 2002)

4 PROPOSTA E SOLUÇÃO DO MODELO

O projeto aqui proposto tem o objetivo de mostrar ao condutor do veículo informações referentes ao funcionamento do sistema de gerenciamento do motor do veículo. A apresentação dessas informações pretende propiciar ao condutor do veículo uma melhoria na condução do veículo utilizado na implementação, que fornece apenas as informações de velocidade, indicador do nível de combustível, indicador da temperatura do líquido de arrefecimento, luz indicadora de pressão do óleo do motor, e hodômetro total. Além disso, o projeto também visa alertar ao motorista se existe algum problema no sistema de gerenciamento do motor, indicando quando possível, o sistema ou componente com defeito.

Para atingir o objetivo almejado, foi definido o seguinte modelo de implementação, apresentado na figura 4.1:

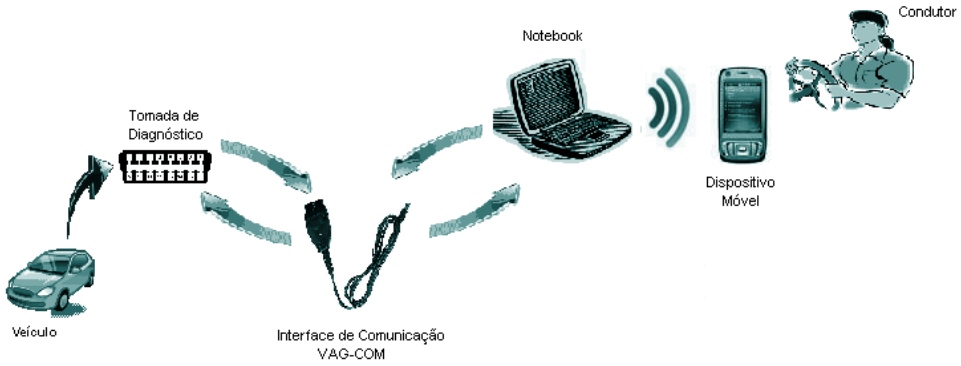

Figura 4.1. Topologia do Sistema de Monitoramento Veicular

4.1 Veículo Utilizado

O veículo utilizado para o projeto é o Gol, ano 97 modelo 98, produzido pela montadora Volkswagen, conforme figura 4.2. O veículo possui o motor AT 1.0 de 8 válvulas e utiliza como combustível apenas gasolina e possui sistema de injeção eletrônica de combustível multiponto.

Figura 4.2. Veículo Volkswagen Gol utilizado no projeto. (Fonte: autor)

Tabela 4.1. Ficha técnica do veículo. (Fonte: http://revistaautoesporte.globo.com/EditoraGlobo/componentes/article/edg_article_print/1,3916,406657-1683-1,00.html, Com adaptações)

Gol 1.0 8v	
Potência (cv/rpm)	54,4/5.000
Torque (Kgfm/rpm)	8,5/3.5000
Velocidade Máxima (Km/h)	147
Aceleração 0-100 Km/h (s)	17,5
Consumo cidade (Km/l)	11,2
Consumo estrada (Km/l)	14,2
Peso (Kg)	935
Tanque de combustível (l)	51

O veículo foi escolhido por que foi nele que foi identificada a ausência de informações importantes de bordo e de diagnóstico do veículo, o que dificulta um monitoramento adequado das condições de funcionamento e condução do veículo. O painel do veículo pode ser visualizado na figura 4.3.

Figura 4.3. Painel de instrumentos do Gol. (Fonte: autor)

O sistema de gerenciamento do motor do automóvel apresentado é o Motronic MP9.0, produzido pela empresa BOSCH e apresentado na figura 4.4. Esse sistema gerencia os sensores e atuadores do sistema de injeção eletrônica do veículo.

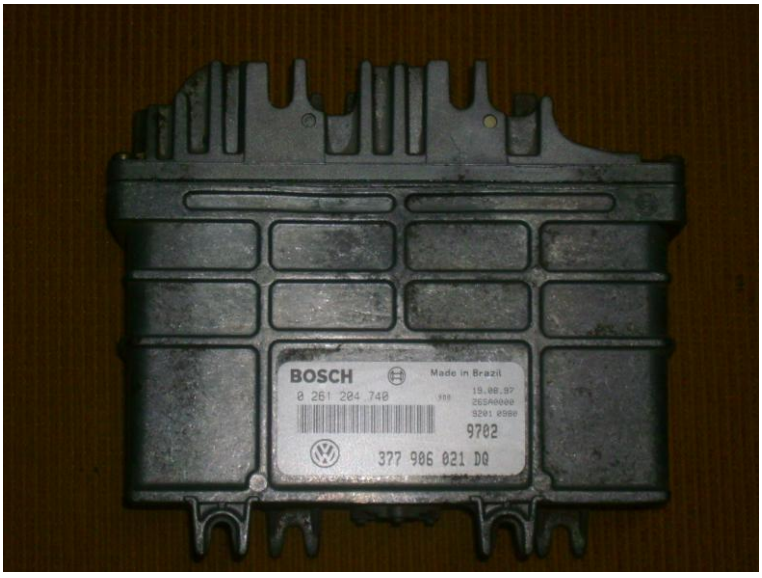

Figura 4.4. UCE Motronic MP9.0. (Fonte: autor)

A UCE em questão possui uma tomada de diagnóstico para a aquisição de informação. Esta tomada utiliza o padrão físico OBDII, e fisicamente é baseado no padrão ISO 9141-2. O padrão possui 16 pinos para alimentação do dispositivo tester e para comunicação. Neste veículo em particular, conforme a figura 4.5, a tomada de diagnóstico possui apenas 3 destes pinos conectados a UCE, os pinos são o pino 7, conhecido como K-line, que é responsável pela comunicação com o dispositivo externo, o pino 4 que é o pino que contém a conexão GND, e o pino 16 que está conectado a alimentação de 12 V corrente contínua para alimentação do dispositivo externo.

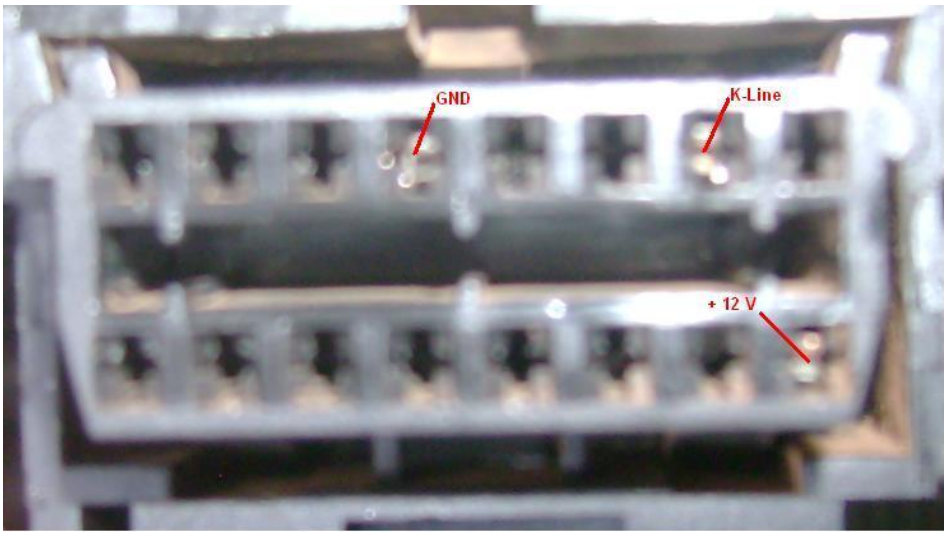

Figura 4.5. Conector OBDII do Gol. (Fonte: autor)

4.2 Interface VAG-COM

Para que a comunicação entre o notebook e a tomada de diagnóstico possa ser realizada, é necessária a utilização de uma interface para a atenuação do sinal de saída da tomada de diagnóstico, que não trabalha com os mesmos valores de tensão e corrente que os computadores padrão PC x386.

Pretendia-se elaborar o hardware da interface, utilizando o CI ELM 327, este CI (Circuito Integrado) tem a função de se comunicar com vários protocolos de comunicação de diagnósticos diferentes, todos baseadas no conector padrão OBDII. Através do estudo do

conector do veículo e sua pinagem, foi identificado através de referências em sites e livros como o do Alexandre Guimarães, que este conector era compatível com o protocolo ISO 9141-2 no qual o CI ELM 327 era compatível.

O CI foi adquirido e a interface de hardware foi elaborada e testada fora do veículo, e constatou-se que todos os valores apresentaram os valores de tensão e corrente compatíveis com os descritos no datasheet do CI. Depois foram efetuados testes no veículo, e durante os testes não foi possível obter qualquer tipo de comunicação com o veículo.

Após inúmeros testes e pesquisas, constatou-se a inviabilidade de utilização desta interface baseada no ELM 327, pois apesar de toda literatura indicar que o tipo de conector do veículo é compatível com o protocolo 9141-2, ele utiliza um protocolo proprietário da Volkswagen, o *Keyword Protocol 1281*, que apesar de ser compatível fisicamente com o protocolo ISO 9141-2, não compartilha as mesmas características de comunicação. Por essa razão foi necessário encontrar outra interface de comunicação, que fosse compatível com o protocolo kw1281**.** Este processo consumiu um tempo considerável da elaboração do projeto, cerca de um mês.

A interface escolhida foi a VAG-COM, por que é uma das poucas que se comunica com o protocolo proprietário KW1281, outro fator relevante para a utilização da interface foi a razoável facilidade para a aquisição deste produto.

Figura 4.6. Interface VAG-COM. (Fonte: autor)

Esta interface, visualizada na figura 4.6, é produzida pela empresa *Ross-Tech*, que está sediada na cidade de *Lansdale, Pennsylvania*, nos Estados Unidos. A *Roos-Tech* é uma

empresa que produz software e hardware de diagnóstico para automóveis. Para a elaboração deste projeto além da interface, foi utilizado o software desenvolvido por esta empresa. O *software* é o VAG-COM, versão 311-2, conforme figuras 4.7 e 4.8. O *software* é uma versão *shareware* com algumas funcionalidades limitadas de diagnostico do veículo, esta versão permite visualizar as informações de bordo e a visualização de alguns erros do sistema de gerenciamento do motor do veículo. O software foi utilizado como modelo e também foi feita a tentativa de engenharia reversa no processo de comunicação com a UCE, através do monitoramento da porta serial do notebook.

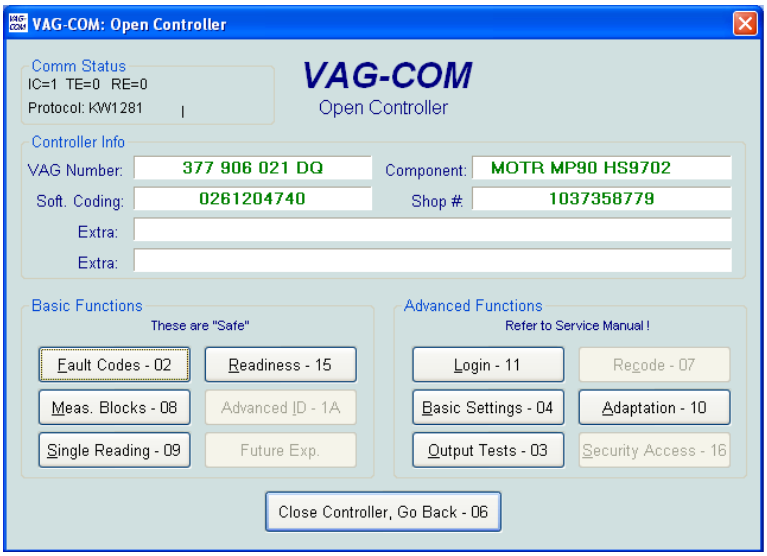

Figura 4.7. Aplicação VAG-COM (exibição dos dados de identificação da UCE). (Fonte: autor)

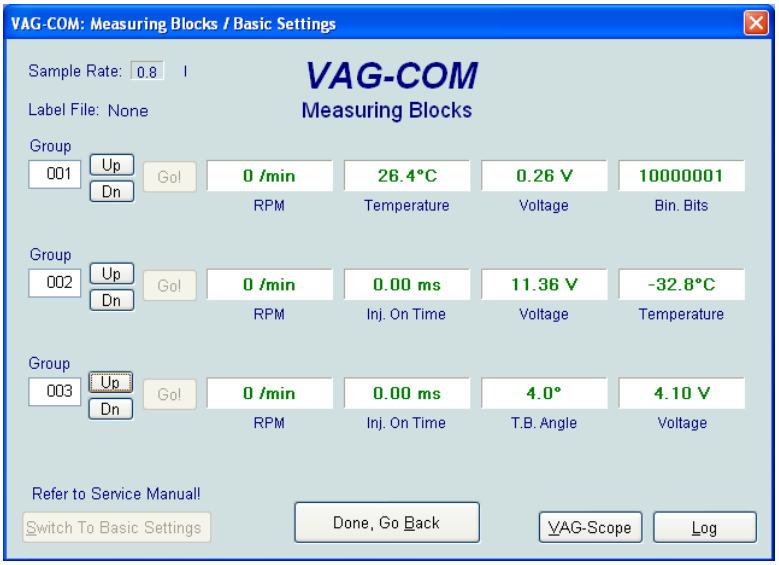

Figura 4.8. Aplicação VAG-COM (valores dos blocos de dados). (Fonte: autor)

Uma característica que merece atenção desta interface é que ela apenas atenua o sinal e faz a conversão do padrão de comunicação para USB. Portanto todo controle do fluxo da informação com a UCE fica a cargo do software desenvolvido, o que demanda um entendimento mais aprofundado do funcionamento do protocolo.

4.3 Interface de Integração

Para a comunicação entre o veículo e o dispositivo móvel, previa-se inicialmente apenas a utilização de uma interface de hardware. Durante a pesquisa e implementação do software no dispositivo móvel e testes da interface notou-se que a interface de comunicação necessitaria de um dispositivo que possuísse o *USB host controller*.

Para alcançar o objetivo do projeto, adotou-se outra solução. Essa solução utiliza um notebook junto à interface VAG-COM como mediador da comunicação entre a UCE do veículo e o dispositivo móvel. A interface VAG-COM atenua o sinal oriundo da UCE do veículo coletado na tomada de diagnóstico e o repassa a porta USB do notebook, que possui instalado o *driver* que possibilita a comunicação com a interface de diagnóstico VAG-COM, esse *driver* é fornecido pela empresa FTDI- *Future Technology Devices International Limited*, que produz o *CI* que está embarcado no VAG-COM e faz a conversão do sinal para o padrão USB.

O *notebook* acessa os dados da interface por meio de uma porta de comunicação serial, esta porta é criada por meio do driver da FTDI. A porta serial permite o envio e recebimento de dados da UCE do veículo.

Para estabelecer a comunicação com a UCE do veículo e o *notebook* foi desenvolvido um software embarcado no *notebook*. Este software estabelece comunicação com a UCE do veículo, e coleta os dados disponíveis no canal de comunicação de forma síncrona. Após a coleta os dados são tratados, onde é feita a identificação da informação coletada e seu tratamento para posterior envio ao dispositivo móvel.

Após o tratamento da informação ela é enviada por meio de transmissão de dados Bluetooth para o dispositivo móvel. Para tanto, é utilizado um adaptador Bluetooth.

4.4 Dispositivo Móvel

O equipamento escolhido para a apresentação das informações para o usuário foi o dispositivo móvel smartphone da marca HTC, modelo TYTNII.

Tabela 4.2. Especificação do TYTNII (com adaptações). (Fonte: http://www.htc.com/www/product/tytnii/specification.html. Com adaptações.)

TYTN II	
Processor	Qualcomm® MSM7200TM, 400MHz
Operating System	Windows Mobile® 6 Professional
Memory	ROM: 256MB
	RAM: 128MB SDRAM
Dimension	112 mm (L) X 59 mm (W) X 19 mm (T)
Weight	190g with battery
Display	2.8 inch, 240 X 320 QVGA TFT-LCD display with adjustable angle and backlight
Connectivity	Bluetooth® 2.0
	Wi-Fi®: IEEE 802.11 b/g
	HTC ExtUSB™ (11-pin mini-USB and audio jack in one)
	GPS antenna connector
Battery	1,350 mAh rechargeable Li-polymer battery Standby time:
	• Up to 350 hours for UMTS
	• Up to 365 hours for GSM
	Talk time:
	• Up to 264 minutes for UMTS
	• Up to 420 minutes for GSM
	• Up to 120 minutes for video call
AC Adapter	Voltage range/frequency: 100 ~ 240V AC, 50/60Hz
	DC output: 5V and 1A

Este aparelho foi escolhido por sua capacidade relativamente alta de processamento para um dispositivo móvel, possui uma tela com tamanho adequado para visualização da informação pelo condutor do veículo, sem prejudicar a visibilidade original do veículo e minimizando a perda de atenção do motorista, e por ser um dispositivo compacto, pode ser embarcado com facilidade no painel do veículo, em uma posição de fácil leitura para o condutor do automóvel.

Além das características citadas anteriormente, o HTC TYTNII utiliza o sistema operacional Windows Mobile 6.0, que é um sistema operacional muito intuitivo e de fácil manuseio pelo usuário, o que facilita a leitura das informações e o manuseio do software

embarcado no dispositivo. Outra característica do Windows Mobile é sua integração nativa com as linguagens da plataforma de desenvolvimento .NET. O TYTNII é apresentado na figura 4.9.

Figura 4.9. HTC TYTNII. (Fonte: http://www.itreviews.co.uk/hardware/h1387.htm)

4.5 O Software Desenvolvido

O sistema desenvolvido neste projeto é constituído de dois softwares que trabalham de forma integrada. O primeiro software está alocado no *notebook* e é responsável por estabelecer uma conexão com a UCE, solicitar as suas informações, coletar os dados disponíveis, identificar e tratar essa informação e disponibilizá-la para o outro software, que está embarcado no dispositivo móvel. Este software por sua vez, recebe a informação já padronizada e a disponibiliza para o condutor do veículo através de uma interface gráfica.

O software embarcado no notebook do veículo é chamado de software de conexão e integração Bluetooth, pois ele está encarregado de se conectar a UCE e transmitir os dados ao dispositivo móvel, através de uma conexão Bluetooth. O software que está embarcado no dispositivo móvel e tem a função de apresentar as informações padronizadas ao condutor do veículo é chamado de software de apresentação da informação ao condutor. Na figura 4.10, pode ser vita a visão geral do sistema.

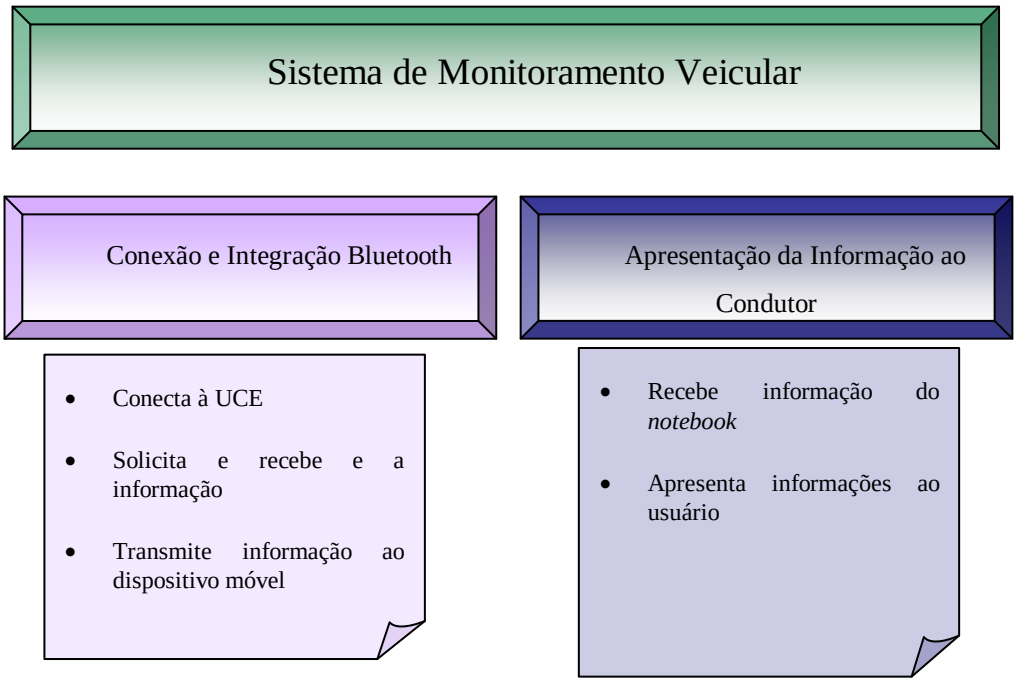

Figura 4.10. Visão geral da organização do software. (Fonte: autor)

4.5.1 Software de Conexão e Integração Bluetooth

O Software de Conexão e Integração Bluetooth é a aplicação integrante do sistema de monitoramento veicular que tem a função de intermediar a comunicação entre o veículo e o dispositivo móvel. A necessidade deste intermédio justifica-se pela ausência de hardware controlador de *host* USB no dispositivo móvel.

Este software, desenvolvido em C#, utiliza a metodologia de linguagem orientada a objeto, e é responsável por coletar e transmitir as informações de bordo e de diagnóstico a UCE do veículo. Para isso o software foi elaborado em 4 classes, onde 3 representam camadas lógicas, e uma classe estática auxiliar. O diagrama de classes é apresentado na figura 4.11.

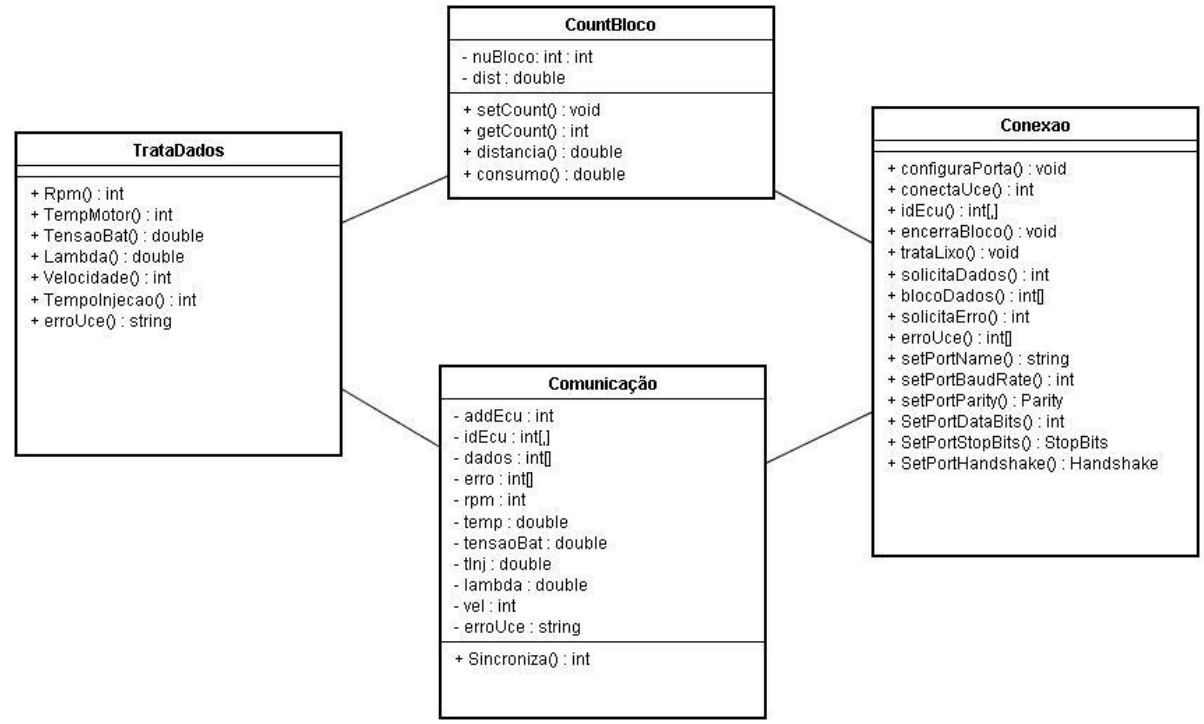

Figura 4.11. Diagrama de Classe da Aplicação. (Fonte: autor)

4.5.1.1 A Classe Conexão

A classe conexão é responsável por estabelecer a conexão com a UCE do veículo, através da interface VAG-COM. Esta conexão utiliza o padrão serial de comunicação para se comunicar com a interface VAG-COM, que por sua vez repassa o sinal a UCE do veículo.

Essa classe contém os métodos necessários para a efetivação da conexão com a UCE, e para o controle da comunicação.

4.5.1.2 A Classe TrataDados

Essa classe é responsável por receber os bytes recebidos da classe Conexão e computar esses valores para obter a informação de bordo e diagnóstico no padrão convencional de leitura. Ela utiliza a camada de conexão com base para a aquisição dos dados, e de acordo com a informação obtida é aplicado o método adequado na camada de comunicação.

Na tabela 4.3 são mostrados os cálculos utilizados na implementação do protótipo. Cada valor é composto de 3 bytes, um byte identifica o tipo de informação e os dois restantes (chamados de byte **a** e byte **b**) são manipulados através de cálculos para a apresentação do valor padronizado ao condutor do veículo.

Tabela 4.3. Cálculos de conversão das informações de bordo. (Fonte: autor)

Valor	Cálculo *(Fonte: http://www.blafusel.de/)
RPM	0.2 * a * b
Tempo de Injeção	a * b * 0.01
Tensão da Bateria	0.001*a*b
Temperatura do Líquido de Arrefecimento do Motor	a * (b - 100) * 0.1
Velocidade	0.01*a*b
Relação Lambda	0.0001 * a * (b - 128) + 1

4.5.1.3 A classe Comunicação

Esta classe é responsável por instanciar as classes anteriores de forma ordenada para que os dados possam ser obtidos. Esta classe também é responsável por estabelecer a conexão e envio de dados ao dispositivo móvel de forma sincronizada.

É na classe de comunicação que são solicitadas através de dados as informações de bordo e de diagnóstico da UCE do veículo. As informações de bordo são solicitadas através dos métodos solicitaDados e blocoDados.

O método solicitaDados, solicita à UCE informações de grupo de leitura de dados que são as informações de bordo, além de requisitar o grupo de dados este método também informa qual bloco de dados que será transmitido pela UCE. Cada bloco de dados contém quatro valores de bordo do veículo.

Tabela 4.4. Blocos de dados e valores identificados. (Fonte: autor)

Bloco de Dados	Valores por grupo
1	RPM, Temperatura do líquido de arrefecimento do motor,
2	RPM, Tempo de injeção, Tensão da bateria, temperatura do ar admitido
3	RPM, Tempo de injeção, angulação da borboleta de aceleração
4	RPM, Tempo de injeção e velocidade
5	RPM, fator lambda
6	Fator lambda, Fator lambda
7	RPM, Temperatura do líquido de arrefecimento do motor
9	Fator lambda

Pode-se notar que o valor de RPM aparece de forma redundante em vários blocos, isso é útil na aplicação, pois este campo é atualizado mais vezes que os outros campos.

As informações apresentadas ao condutor foram selecionadas de acordo com sua relevância e facilidade de entendimento, então foi dada prioridade nas informações que o condutor geralmente já tem familiaridade.

As informações de erro de algum componente da UCE são adquiridas dos métodos solicitaErro e erroUce. O método solicitaErro faz a requisição do grupo de leitura de erros, e o método erroUce retorna os erros encontrados, se não houver erro armazenado na memória de erro da UCE, o valor retornado será igual a zero, caso contrário será retornado três bytes para a identificação de cada erro.

Os dois primeiros bytes do erro são concatenados em valor hexadecimal e posteriormente convertido em um valor inteiro, este valor inteiro é comparado com os códigos de erro armazenados em um arquivo texto. Esse arquivo contém vários códigos de erro, com suas respectivas descrições, que são retornadas pelo método erroUce, da classe trataDados. Abaixo, na figuara 4.12 é visulizado o diagrama de caso de uso na visão da aplicação.

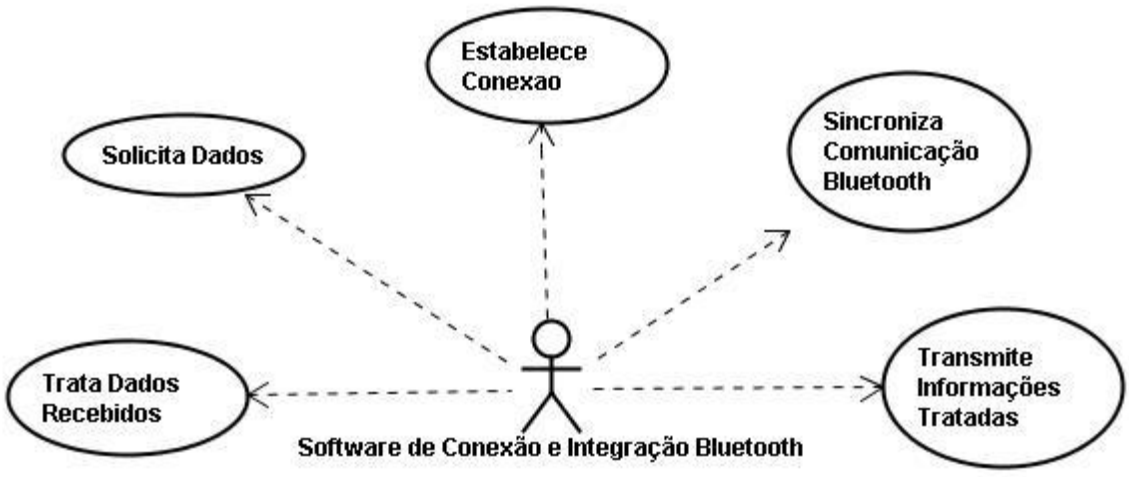

Figura 4.12. Diagrama de Caso de Uso da Aplicação. (Fonte: autor)

4.5.2 Software de Apresentação da Informação ao Condutor

O software embarcado em dispositivo móvel é responsável por receber os dados transmitidos pela aplicação embarcada em notebook e apresentar essa informação ao usuário do veículo. Para isso o sistema possui duas classes que adquirem a informação e apresentam esta informação ao condutor do veículo através de um formulário padrão do Windows Mobile.

Conforme a figura 4.13, o software é composto de duas classes: a classe Comm e a classe FormApresentacao.

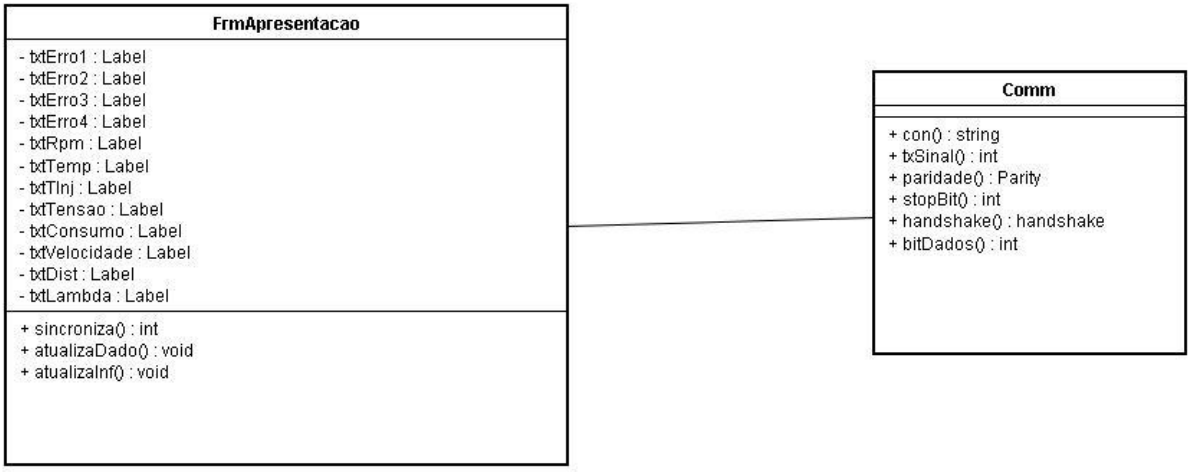

Figura 4.13. Diagrama de classe da aplicação embarcada no dispositivo móvel. (Fonte: autor)

4.5.3 A Classe Comm

Esta classe é responsável pelas configurações da comunicação serial Bluetooth. No programa a comunicação é feita de forma serial através de uma porta COM, isso proporciona uma transparência da implementação da comunicação Bluetooth entre o dispositivo móvel e o *notebook*

4.5.4 A Classe FrmApresentação

Esta classe é responsável pelo controle do formulário que apresenta graficamente as informações ao condutor do veículo, e também é responsável por coletar as informações transmitidas pelo notebook.

Esta Classe possui a implementação de uma Thread. A Thread é necessária, pois existe concorrência de processamento entre a execução e controle do formulário e a coleta e atualização dos dados no formulário, e a atualização destas informações só é possível com este tipo de processamento paralelo.

As informações são apresentadas em três guias, duas são destinadas a informar as informações de condução e a restante, as informações de diagnóstico.

Figura 4.14. Guias de apresentação das informações da aplicação. (Fonte: autor)

Pode-se notar na figura 4.14 que são apresentadas as seguintes informações:

- Velocidade
- Rotação do motor
- Temperatura do líquido de arrefecimento
- Distância Percorrida
- Tensão da bateria
- Consumo de combustível
- Relação lambda;
- Tempo de injeção;

- Erros apresentados 

A aplicação de apresentação de informações ao condutor contém dois usuários, a própria aplicação que é usuária da aplicação embarcada no notebook e o condutor do veículo, que responsável pela inicialização e finalização da informação, além da escolha da guia de informação a ser exibida, essas ações são mostradas na figura 4.15 e 4.16.

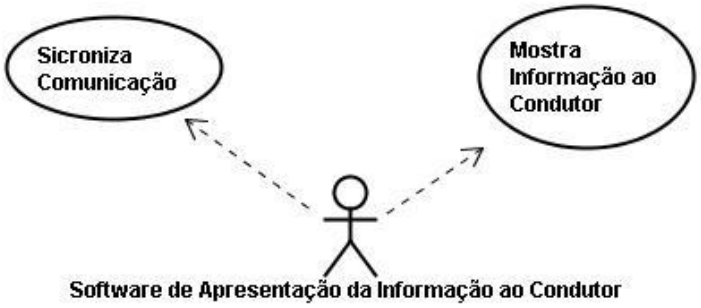

Figura 4.15. Diagrama de Caso de Uso (Visão do sistema). (Fonte: autor)

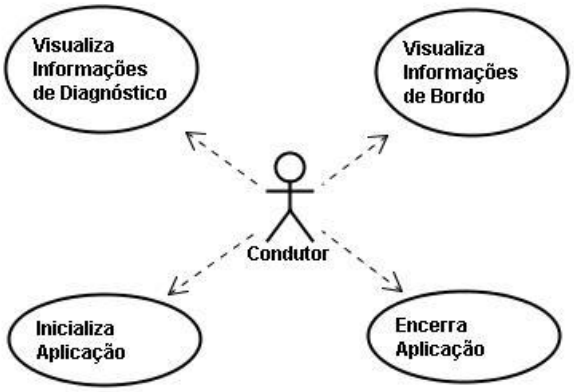

Figura 4.16. Diagrama de Caso de Uso (Visão do Condutor). (Fonte: autor)

5 APLICAÇÃO DA SOLUÇÃO COM RESULTADOS

Após a conclusão das funcionalidades básicas do software embarcado no notebook e no dispositivo móvel, iniciaram-se os testes no veículo. Os primeiros testes ocorreram com o veículo em repouso e com o motor desligado. Através deste teste foi possível depurar a aplicação para a identificação das informações recebidas e verificar a precisão dos resultados obtidos, através da comparação com os dados disponíveis no veículo e com os dados obtidos da aplicação VAG-COM. O software ficou em estado de execução por cerca de 50 minutos ininterruptos, sem perca de informação, conforme a fugura 5.1.

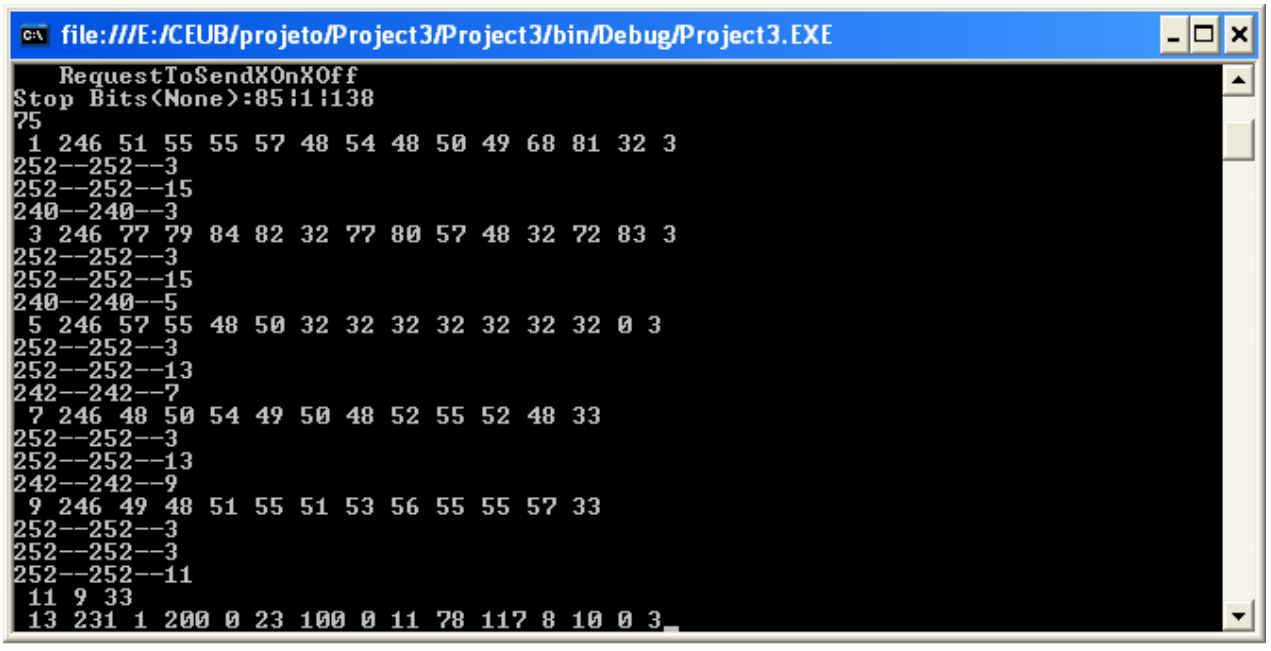

Figura 5.1 Dados brutos recebidos da UCE do veículo. (Fonte: autor)

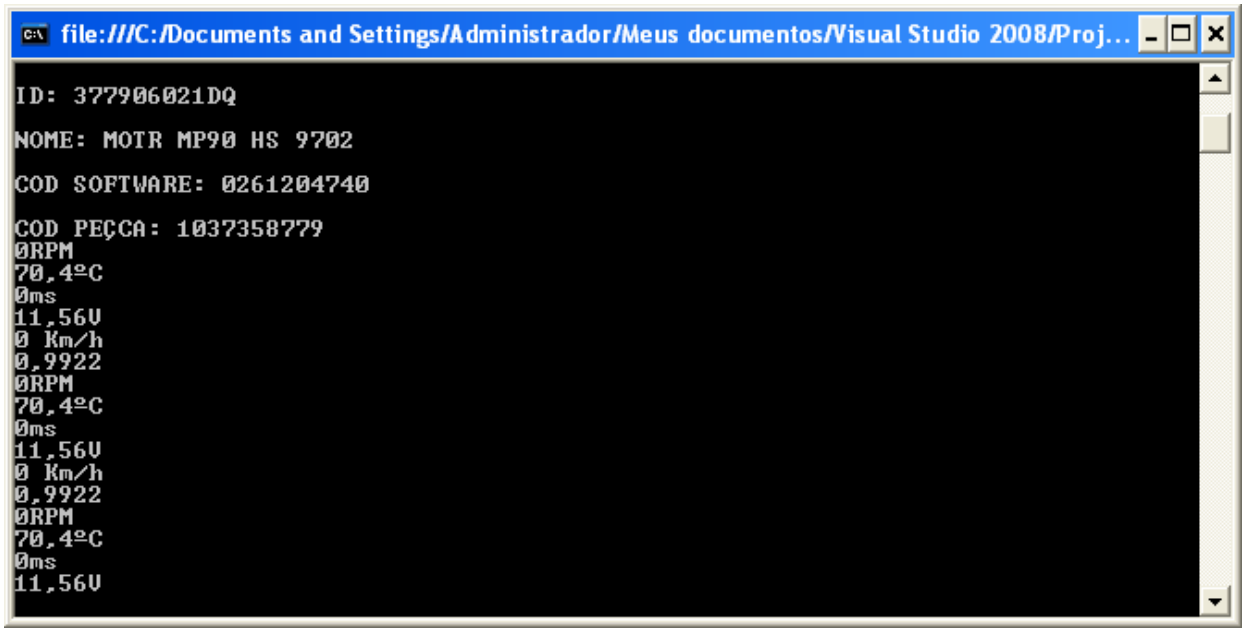

Figura 5.2. Dados recebidos da UCE tratados. (Fonte: autor)

Pode-se notar nas figuras 5.3 e 5.4 o processo de conversão dos bytes recebidos em valores inteligíveis pelo condutor.

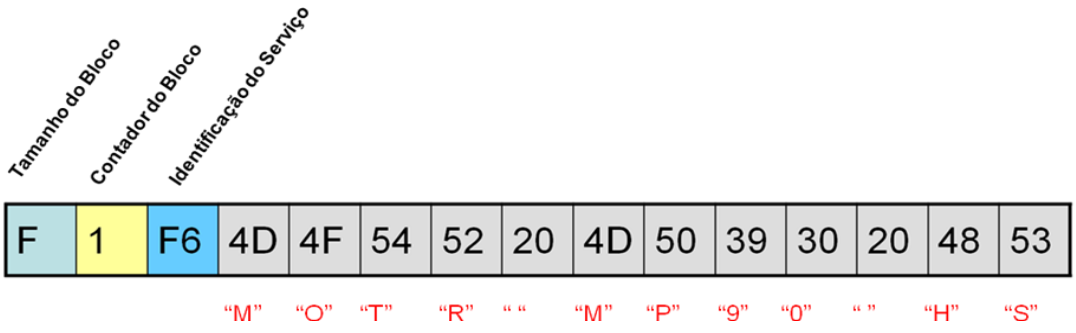

Figura 5.3. Exemplo de conversão do bloco de identificação. (Fonte: autor)

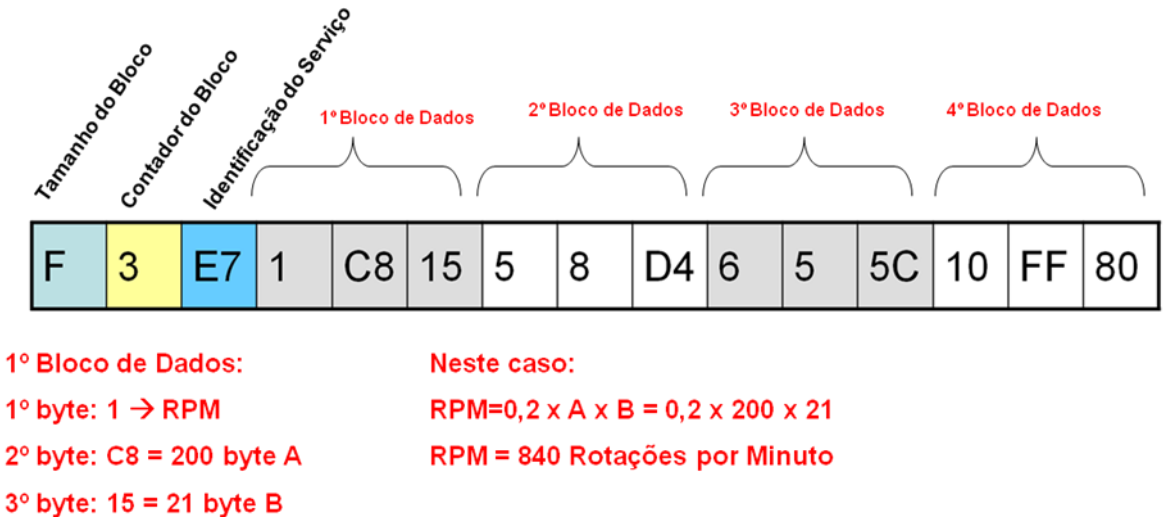

Figura 5.4. Exemplo de Conversão do bloco de dados de bordo. (Fonte: autor)

Com a obtenção e tratamento dos dados realizados com sucesso, foram feitos testes com o carro em repouso, mas com o motor em funcionamento, e os resultados foram comparados com os valores do painel de instrumentos do veículo e com os valores disponibilizados no *software* VAG-COM. O resultado pode ser visto na figura 5.5.

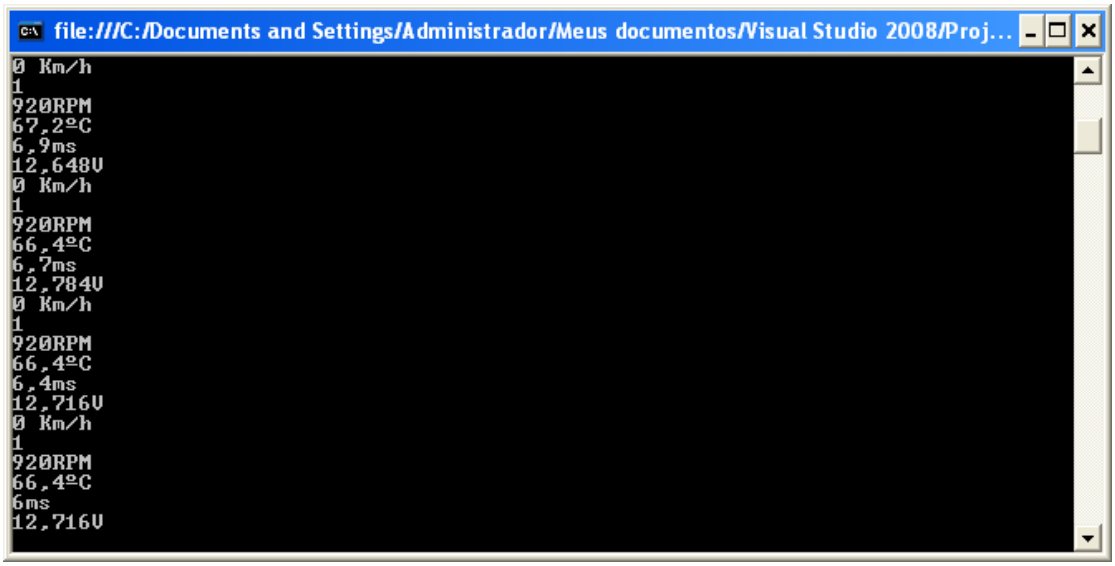

Figura 5.5. Dados recebidos da UCE tratados com o motor do veículo ligado (Fonte: autor)

Após a conclusão do teste com o motor ligado, foi feita a integração da comunicação entre a aplicação do notebook e o dispositivo móvel, como pode ser visto na figura 5.6, com o fornecimento das informações do bordo.

Figura 5.6 Dados recebidos pelo dispositivo móvel e mostrados em aplicação do dispositivo móvel. (Fonte: autor)

Após os testes com o veículo em repouso, iniciaram-se os testes com o veículo em movimento. O primeiro teste consistente com o veículo em movimento foi realizado no dia 28/10/2009, na rodovia BR-070, em um percurso de cerca de 25 quilômetros Neste percurso realizou-se diversas repetições de testes de funcionamento, neste primeiro teste foram mostradas ao condutor do veículo a velocidade, rotação do motor, temperatura do líquido de arrefecimento e tensão da bateria.

Os valores citados acima apresentaram as informações consistentemente ao condutor do veículo. Os valores de velocidade e temperatura do líquido de arrefecimento quando comparados aos valores fornecidos pelo painel de instrumentos mostraram-se bastante precisos, ao valor muito próximo ao apresentado no painel. Mas a atualização do valor da velocidade no dispositivo móvel apresentou um atraso na disponibilização da informação, quando se aplicava alta carga ao motor, provocando uma variação rápida de velocidade.

Durante este teste foi identificado que o sistema interrompia seu funcionamento constantemente antes dos 5 minutos de funcionamento. Após alguns testes foi identificado que ao requisitar o bloco de dados de bordo 0x05 da UCE, o mesmo funcionava e apresentava os resultados esperados com o veículo em repouso, mas quando o veículo se movimentava, em pouco tempo a comunicação se perdia. Quando se solicitou outro bloco de informação com a mesma informação a comunicação persistiu em funcionamento.

Foi feito um percurso na rodovia BR 070 do quilômetro por cerca de 20 quilômetros, e notou-se que após não solicitar mais o bloco de dados 0x05 a comunicação manteve-se ininterrupta durante todo trajeto, apresentado as informações como desejado.

Depois do teste com o veículo em movimento em rodovia, foi feito um refinamento do software embarcado em notebook e no do dispositivo móvel, para a automatização da sincronização dos equipamentos, e também para fornecer todas as informações ao condutor do veículo.

Para a disponibilização da relação lambda que era adquirida no bloco 5, foi utilizado o bloco 9 para a aquisição dessa informação, após teste de percurso de 12 km, o software apresentou as informações ininterruptamente. Além disso, este bloco é apresentado alternadamente ao bloco 2 que fornece informação redundante de RPM, tempo de injeção e tensão da bateria, uma vez que as informações são redundantes e a tensão da bateria não necessita de uma alta taxa de atualização. Utilizou-se este recurso para a requisição de 3 blocos de dados, o que minimiza o atraso da apresentação das informações ao condutor do veículo.

Para disponibilizar as informações de distância percorrida e consumo de combustível é necessária a implementação de alguns cálculos, pois essas informações não são nativas da UCE do veículo. Para o cálculo da distância percorrida decidiu-se utilizar o parâmetro de velocidade como base para o cálculo. Para informar a distância utiliza-se a velocidade percorrida em um intervalo de tempo, este intervalo é constante, e é baseado no tempo de fornecimento da informação da velocidade do veículo. Ou seja, a cada intervalo de tempo será adquirida a velocidade do veículo e a distância será calculada e armazenada em uma variável acumuladora.

$$distância = distância\_anterior + velocidade \cdot tempo(consante) \quad \text{(Eq. 5. 1)}$$

Para o cálculo do consumo é necessário utilizar o valor do tempo de injeção fornecido pela UCE do veículo e da vazão do bico injetor, este valor foi baseado na especificação de fornecedoras de autopeças.

$$Volume = Volume\_anterior + tempo\_de\_injeção * vazão\_bico\_injetor \qquad \text{(Eq. 5. 2)}$$

$$Consumo = \frac{Distância}{Volume} \qquad \text{(Eq. 5. 3)}$$

Após a implementação destas informações, efetuou-se um teste com o veículo em movimento, neste teste foi ajustado o valor da constante de tempo usada no cálculo da distância, obtendo uma precisão de cerca de 85 metros. O consumo calculado apresentou valores razoáveis e dentro da realidade, entre 6 km/l e 14 km/l. Como o valor é instantâneo não se pode se comparar tal valor com o consumo médio. Para o valor do consumo de combustível apresentar valores aceitáveis foi necessário que o veículo percorresse uma distância superior a 8 km. Os resultados obitidos podem ser observados na figura 5.7.

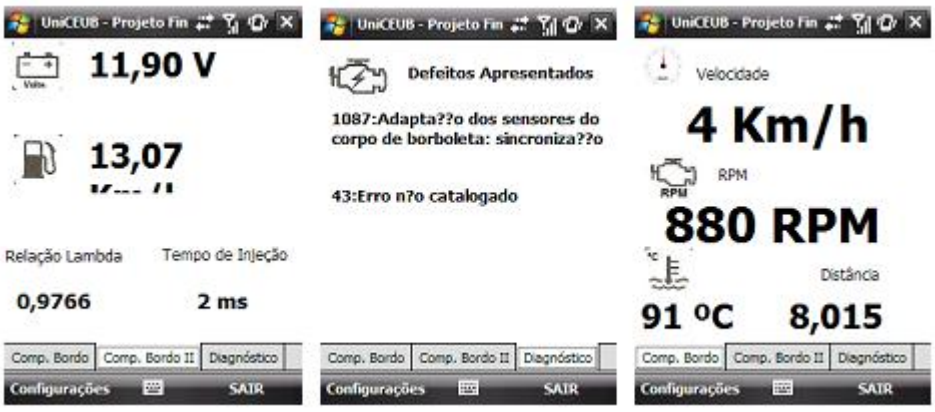

Figura 5.7. Resultados obtidos durante os testes. (Fonte: autor)

Por último foram feitos os ajustes de formatação da informação apresentada no dispositivo móvel para acomodar os valores de uma forma mais apresentável.

5.1 Avaliação Global da Solução Proposta

De forma geral, a implementação do projeto apresentou resultados satisfatórios. Foi possível se conectar à UCE do veículo, solicitar dados, calcular e converter valores, e disponibilizar as informações ao condutor do veículo. Como a maior parte das informações é adquirida da UCE do veículo elas mostram o valor real interpretado pelo módulo.

Em relação às informações de distância percorrida e consumo, que são calculadas com base nas informações obtidas os resultados foram relativamente precisos. À distância percorrida se mostrou condizente com a informação do hodômetro total do veículo e com testes feitos entre dois pontos com distância conhecida.

O valor do consumo instantâneo de combustível mostrou durante os primeiros testes variações bruscas do valor e às vezes não condiziam com a realidade. Após tal constatação o calculo foi ajustado e os resultados obtidos se aproximaram da realidade de consumo do veículo.

Durante os testes do projeto com o veículo em movimento notou-se uma menor velocidade de atualização das informações em relação às disponibilizadas no painel de instrumentos do veículo, o que provoca uma defasagem na apresentação da informação em relação ao valor mostrado no painel de instrumentos. Mas mesmo assim é possível utilizá-las como referência para condução.

Notou-se, principalmente no início do projeto uma grande dificuldade de adquirir informações a respeito do assunto aqui tratado e as informações obtidas em referências acadêmicas se mostraram algumas vezes incompletas, pela própria natureza dessas informações, que são informações industriais automobilísticas.

Após pesquisa em diversas fontes, e principalmente na internet foi possível identificar qual o protocolo de diagnóstico da UCE do veículo, e através do site http://www.blafusel.de/, que fornece informações técnicas do protocolo que não estão descritas na norma da SAE que específica o protocolo kw1281. Este site foi a base do desenvolvimento do software e foi fundamental para a conclusão deste projeto.

A autonomia do protótipo está limitada à duração das baterias do dispositivo móvel e do *notebook*, esta autonomia é maior no dispositivo móvel, que após carga apresentou

funcionamento por mais de 6 horas durante os testes. Por outro lado a bateria do notebook não ultrapassou 2 horas e 30 minutos de carga durante os testes.

Durante os testes aconteceram alguns erros esporádicos, alguns identificados e outros não. Após o refinamento do software, durante os testes por um período prolongado, o notebook às vezes “travava”, após tal constatação foram feitos ajustes de configuração e desempenho do notebook, esse procedimento otimizou o desempenho do notebook e minimizou o risco deste tipo de incidente.

Raramente, durante a comunicação a mesma era interrompida pela UCE e não foi identificada a causa deste problema, permanecendo como um erro pontual.

Para a realização dos testes encontrou-se certa dificuldade, pois os testes foram realizados em vias publicas do Distrito Federal, e as condições de tráfego e tempo não são controladas, foi necessário em algumas situações o auxílio de uma pessoa para conduzir o veículo enquanto outra observava os dados adquiridos.

Outra característica do projeto é que o mesmo é um protótipo que tem o objetivo de mostrar a viabilidade de se utilizar as informações da UCE que são fornecidas através da tomada de diagnóstico para fornecer ao condutor do veículo informações do bordo e diagnóstico, e a maioria dessas informações estão ausentes no veículo utilizado. Ou seja, não se pretende neste projeto implementar uma solução comercial de alta qualidade e desempenho.

Para a implementação de um sistema para uso comercial, seria necessário aperfeiçoar o software para a identificação e tratamento dos erros pontuais apresentados, além disso, seria necessário efetuar testes em diversas UCE’s que possuem este protocolo, para analisar se elas possuem as mesmas informações. Além disso, os testes efetuados teriam que ser feitos em ambientes controlados apropriados como campo de provas, e a montadora do veículo teria que homologar tal sistema.

Para a elaboração de projetos futuros, sugere-se a integração dos componentes em um dispositivo único, embarcado ao veículo, de forma a reduzir o custo e complexidade do projeto atual e melhorar a integração do sistema com o veículo. Sugere-se também o estudo de como utilizar as informações provenientes da UCE do veículo de forma consolidada, para produzir indicadores de condução do veículo.

Outra sugestão de projeto futuro é o estudo de viabilidade de atuação na UCE do veículo para implementação de um piloto automático com controle de velocidade ou um sistema de atuação na UCE para evitar colisões.

6 CONCLUSÃO

Os resultados obtidos foram satisfatórios, tanto com o veículo em repouso quanto em movimento. O protótipo desenvolvido mostrou que é possível utilizar as informações de diagnóstico do veículo para o monitoramento do sistema de gerenciamento do motor, fornecendo informações adicionais ao veículo utilizado no projeto.

Para alcançar os objetivos almejados, foi necessário superar algumas dificuldades encontradas no decorrer do desenvolvimento do projeto, que não eram esperadas, como a falta de informação sobre o protocolo de comunicação kw1281, ausência de exemplos de software que realizam esta função, não disponibilização de informações técnicas pela montadora do veículo e pela fabricante da UCE do veículo.

Para projetos futuros é sugerida a implementação do protótipo em um conjunto de hardware/software embarcado ao veículo, através de um hardware eletrônico integrado, de forma a não se utilizar a intermediação de um notebook durante a comunicação. Também pode ser feito um estudo para utilizar as informações adquiridas da UCE e gerar indicadores gráficos e sonoros dinâmicos de condução do veículo, como também a atuação na UCE do veículo, para a implementação de um piloto automático.

REFERÊNCIAS BIBLIOGRÁFICAS

BERNAL, Paulo Sérgio Milano. **Comunicações Móveis Tecnologias e Aplicações.** São Paulo: Érica, 2002.

BOSCH, Robert. **Manual de Tecnologia Automotiva.** São Paulo: Edgar Blucher, 2005.

Brasil. *Lei nº 8.723, de 28 de outubros de 1993*. Dispõe sobre a redução de emissão de poluentes por veículos automotores e dá outras providências, 28 out. 1993. Disponível em: <http://www.planalto.gov.br/ccivil_03/LEIS/L8723.htm>. Acesso em: 02 Out. 2009. 10:00.

Brasil, *Resolução nº 354, de 13 de dezembro de 2004.* Dispõe sobre os requisitos para adoção de sistemas OBD nos veículos automotores leves objetivando preservar a funcionalidade dos sistemas de controle de emissão, 14 dez. 2004. Disponível em: <http://www.ibama.gov.br/proconve/ArquivosUpload/1354.pdf>. Acesso em: 02 Out. 2009. 11:26.

COZENS, Simon, WAINWRIGHT, Peter. **Beginning Perl.** Chigago:Wrox Press, 2000.

GONÇALVES, Majô. **Pesquisa Revela Números sobre Manutenção Preventiva.** Disponível em < http://www.coisasdeagora.com.br/noticias.asp?id=260>. Acesso em: 16 Set 2009. 09:43

GUIMARÃES, Alexandre. **Eletrônica Embarcada Automotiva.** São Paulo: Editora Érica Ltda. 2007.

KIENCKE, Uwe, NIELSEN, Lars. **Automotive Control Systems: For Engine, Driveline, and Vehicle.** Illus: Springe. 2005.

Society of Automotive Engineers. **Keyword Protocol 1281.** USA: SAE. 2008.

STONE, Richard. **Introduction to Internal Combustion Engines.**USA**:** SAE, 1999.

APÊNDICE A – CÓDIGO FONTE DA APLICAÇÃO EMBARCADA NO NOTEBOOK- CONEXÃO E INTEGRAÇÃO BLUETOOTH

Classe Conexao

```
using System;
using System.Collections.Generic;
using System.Text;
using System.IO;
using System.IO.Ports;

namespace Projeto_Final
{
    class Conexao : System.IO.Ports.SerialPort
    {


 Inicialização da UCE - É enviado um sinal de 5 baud (simulado em uma conexão de 9600 bps
através de timers
            // de 200ms entre os sinais) e com o parâmetro 01(endereço da UCE)


            public void configuraPorta(SerialPort rs) {
                rs.PortName = SetPortName(rs.PortName);        // configura a porta COM a ser
usada - COM1
                rs.BaudRate = SetPortBaudRate(rs.BaudRate);    // configura a taxa de
sinalização da conexão
                rs.Parity = SetPortParity(rs.Parity);          // configura a paridade de bits
                rs.DataBits = SetPortDataBits(rs.DataBits);    // configura a quantidade de
bits de dados
                rs.Handshake = SetPortHandshake(rs.Handshake); // configura o handshake da
conexão

            }
            public int conectaUce(SerialPort rs){

                int sync=0;     //variável que recebe o byte de sincronização
                int kw1=0;      //variável que recebe o keyword1, byte menos significativo,
0X01
                int kw2=0;      //variável que recebe o keyword2, byte mais significativo,
0X8A

                rs.BreakState = true; //bit 0
                System.Threading.Thread.Sleep(200);   //timer para simular comunicação de 5
baud
                rs.DtrEnable = false;                 //habilita a opção de dado para
transmitir
                rs.BreakState = false;                //bit 1
                System.Threading.Thread.Sleep(200);
                rs.BreakState = true;                 //bit 0
                System.Threading.Thread.Sleep(200);
                rs.BreakState = true;                 //bit 0
                System.Threading.Thread.Sleep(200);
                rs.BreakState = true;                 //bit 0
```

```
                System.Threading.Thread.Sleep(200);
                rs.BreakState = true;                   //bit 0
                System.Threading.Thread.Sleep(200);
                rs.BreakState = true;                   //bit 0
                System.Threading.Thread.Sleep(200);
                rs.BreakState = true;                   //bit 0
                System.Threading.Thread.Sleep(200);
                rs.BreakState = true;                   //bit 0
                System.Threading.Thread.Sleep(200);
                rs.BreakState = false;                  //bit 1
                System.Threading.Thread.Sleep(200);
                rs.DtrEnable = true;                   //Habilita a opção para recepção de
dados, neste caso da ECU
                System.Threading.Thread.Sleep(200);
                //Rotina para a recepção dos dados esperados da ECU, que confirmam que o
processo de
                //inicialização da UCE ocorreu com Sucesso
                rs.ReadByte();       //recebe informação não necessária para a aplicação
                rs.ReadByte();       //recebe informação não necessária para a aplicação
                sync = rs.ReadByte();
                kw1 = rs.ReadByte();
                kw2 = rs.ReadByte();
                rs.ReadByte();
                rs.ReadByte();
                rs.ReadByte();
                if (sync == 0x55 && kw1 == 0x01 && kw2 == 0x8A){
                    return kw1;
                }else
                    return 0;
                }
//-------------------------------------------------------------------------------------
           public int[,] idEcu(SerialPort rs){

                int resp,j,ctrl,k,l;
                int i=0;
                int[,] idEcu=new int[5,16];  //array para retornar o id da ECU
                byte[] dado=new byte[1];       //variável que envia o byte como complemento a
ECU
                dado[0] = 0x75; //Atrbui o complemeto de 0x8A para envio a a UCE
                rs.Write(dado, 0, 1);               //envia byte de complemento a ECU
                i = 0;
                    while(i<5){

                        rs.ReadByte();
                        resp = rs.ReadByte(); //tamanho do bloco
                        k=0;
                        j = resp;
                        l = j;

                        while (j >0){         //loop para pegar os dados do bloco

                            ctrl = 255 - resp;      //calcula o complemento do dado recebido
a ser enviado pela UCE
                            dado[0] = (byte)ctrl;//trata o complemento para envio a UCE
                            rs.Write(dado, 0, 1);
                            rs.ReadByte();
                            resp = rs.ReadByte();
                            if (j == l) {CountBloco.setCount(1);}       //contator do bloco
                           //verifica se o cabeçalho é o cabeçalho de dados de id
                           if(resp==0xf6){
                                //idEcu[i, k] = i - 1; ;
                                ctrl = 255 - resp;      //calcula o complemento do dado
recebido a ser enviado pela UCE
                                dado[0] = (byte)ctrl;   //trata o complemento para envio a
UCE
                                rs.Write(dado, 0, 1);

                                while(j>=0){
                                       rs.ReadByte();
                                       resp=rs.ReadByte();
                                         if(resp!=0X03){     //verifica se foi recebido o
byte que indica o final do bloco
                                             idEcu[i,k]=resp;      //atribui ao array os bytes
da id da UCE
                                             k++;                  //incrementa a variável que
controla as colunas do array
                                             j--;
                                             ctrl = 255 - resp;      //calcula o complemento
do dado recebido a ser enviado pela UCE
                                             dado[0] = (byte)ctrl;   //trata o complemento
para envio a UCE
```

```
                                        rs.Write(dado, 0, 1);
                                    }else{
                                        encerraBloco(rs); //função pra encerra bloco
                                        break;

                                    }

                                }
                            }
                            j--;
                        }
                        ++i;
                    }

                return idEcu;

            }
//-------------------------------------------------------------------------------------
private void encerraBloco(SerialPort rs){
                //System.Console.Write("\n");
                int resp,j,ctrl;
                byte[] dado=new byte[1];
                dado[0] = 0x03;
                rs.Write(dado, 0, 1);
                rs.ReadByte();
                rs.ReadByte();
                dado[0] =(byte)CountBloco.getCount(); //n° do bloco
                rs.Write(dado, 0, 1);
                rs.ReadByte();
                rs.ReadByte();
                dado[0] = 0x09;
                rs.Write(dado, 0, 1);
                rs.ReadByte();
                rs.ReadByte();
                dado[0] = 0x03;
                rs.Write(dado, 0, 1);
                resp = rs.ReadByte();
                if (resp == 0x03)
                {
                    j = 0;
                    for (j = 0; j < 3; j++)
                    {
                        ctrl = 255 - resp;
                        dado[0] = (byte)ctrl;          // converter int recebido para byte
                        rs.Write(dado, 0, 1);
                        resp = rs.ReadByte();
                    }
                }
            }
            public void trataLixo(SerialPort rs) {
                int resp,ctrl;
                int i=0;
                byte[] dado = new byte[1];

                resp = rs.ReadByte();

                while (i < resp)
                {
                    ctrl = 255 - resp;
                    dado[0] = (byte)ctrl;
                    rs.Write(dado, 0, 1);
                    rs.ReadByte();
                    resp = rs.ReadByte();
                    i++;
                }

                CountBloco.setCount(1);
            }
             //-------------------------------------------------------------------------
            public int solicitaDados(SerialPort rs,int nuSub){

                 byte[]dado=new byte[1];
                 int resp;

                  dado[0] = 0x04;
                  rs.Write(dado, 0, 1);
                  System.Threading.Thread.Sleep(10);
                  rs.ReadByte();
                  System.Threading.Thread.Sleep(10);
                  resp = rs.ReadByte();
```

```
                dado[0] = (byte)CountBloco.getCount();
                rs.Write(dado, 0, 1);
                System.Threading.Thread.Sleep(10);
                rs.ReadByte();
                System.Threading.Thread.Sleep(10);
                rs.ReadByte();
                dado[0] = 0x29;    //solicitação de leitura de grupo
                rs.Write(dado, 0, 1);
                rs.ReadByte();
                System.Threading.Thread.Sleep(10);
                rs.ReadByte();
                dado[0] = (byte)nuSub; //0x01;
                rs.Write(dado, 0, 1);
                System.Threading.Thread.Sleep(10);
                rs.ReadByte();
                rs.ReadByte();
                dado[0] = 0x03;
                rs.Write(dado, 0, 1);
                rs.ReadByte();
                return (1);

            }
            //-------------------------------------------------------------------------

            public int[] blocoDados(SerialPort rs){
                CountBloco.setCount(1);
                int resp, j, ctrl;
                int k=0;
                byte[] dado=new byte[1];
                int[] blocoDados= new int[16];
                rs.ReadByte();
                System.Threading.Thread.Sleep(10);
                resp = rs.ReadByte();
                j = resp;
                while (j > 0)
                {
                    ctrl = 255 - resp;
                    dado[0] = (byte)ctrl;    //converte int recebido para byte
                    rs.Write(dado, 0, 1);
                    System.Threading.Thread.Sleep(10);
                    rs.ReadByte();
                    System.Threading.Thread.Sleep(10);
                    resp = rs.ReadByte();
                    System.Threading.Thread.Sleep(10);
                    if (resp == 0xE7)
                    {

                        ctrl = 255 - resp;  //calcula o complemento do dado recebido a ser
enviado pela UCE
                        dado[0] = (byte)ctrl;   //trata o complemento para envio a UCE
                        rs.Write(dado, 0, 1);
                        System.Threading.Thread.Sleep(10);
                        while (j > 0)
                        {
                            rs.ReadByte();

                            resp = rs.ReadByte();
                            if (resp != 0X03){      //verifica se foi recebido o byte que
indica o final do bloco
                                blocoDados[k] = resp;      //atribui ao array os bytes da id da
UCE
                                k++;                  //incrementa a variável que controla as
colunas do array
                                j--;
                                ctrl = 255 - resp;      //calcula o complemento do dado
recebido a ser enviado pela UCE
                                dado[0] = (byte)ctrl;   //trata o complemento para envio a UCE
                                System.Threading.Thread.Sleep(10);
                                rs.Write(dado, 0, 1);

                            }

                            else
                            {
                                j -= 3;
                            }
                        }
                      j--;
                    }

                }
```

```
          return blocoDados;

        }
    //-------------------------------------------------------------------------------
         public int solicitaErro(SerialPort rs) {
             byte[] dado = new byte[1];
             int resp;

             dado[0] = 0x03;
             rs.Write(dado, 0, 1);
             rs.ReadByte();
             resp = rs.ReadByte();

             dado[0] = (byte)CountBloco.getCount();
             rs.Write(dado, 0, 1);
             rs.ReadByte();
             rs.ReadByte();

             dado[0] = 0x07;    //solicitação de leitura de grupo
             rs.Write(dado, 0, 1);
             rs.ReadByte();
             rs.ReadByte();

             dado[0] = 0x03;
             rs.Write(dado, 0, 1);
             rs.ReadByte();

             return 1;

         }
        //-------------------------------------------------------------------------------
         public int[] erroUce(SerialPort rs) {

             CountBloco.setCount(1);
             int resp, j, ctrl;
             int k=0;
             byte[] dado=new byte[1];
             int[] erroUce= new int[16];
             rs.ReadByte();
             resp = rs.ReadByte();
             j = resp;
             while (j > 0)
             {
                 ctrl = 255 - resp;
                 dado[0] = (byte)ctrl;
                 rs.Write(dado, 0, 1);
                 rs.ReadByte();

                 resp = rs.ReadByte();
                 if (resp == 0xFC)
                 {

                     ctrl = 255 - resp;      //calcula o complemento do dado recebido a ser
enviado pela UCE
                     dado[0] = (byte)ctrl;   //trata o complemento para envio a UCE
                     rs.Write(dado, 0, 1);
                     while (j > 0)
                     {
                         rs.ReadByte();

                         resp = rs.ReadByte();

                         if (resp != 0X03){      //verifica se foi recebido o byte que
indica o final do bloco
                             erroUce[k] = resp;      //atribui ao array os bytes da id da
UCE
                             k++;                  //incrementa a variável que controla as
colunas do array
                             j--;
                             ctrl = 255 - resp;      //calcula o complemento do dado
recebido a ser enviado pela UCE
                             dado[0] = (byte)ctrl;   //trata o complemento para envio a UCE
                             rs.Write(dado, 0, 1);
                         }

                         else
                         {
                             j -= 3;

                         }
```

```
                    }
                    j--;


                }

            }

          return erroUce;

        }

        //---------------------------------------------------------------------------
//Funções baseadas ma referência da Microsoft sobre portas serias no site da MSDN
(http://msdn.microsoft.com/pt-br/library/system.io.ports.serialport(VS.80).aspx)
        public static string SetPortName(string defaultPortName)
        {
            string portName;

            portName = "COM1";

            if (portName == "")
            {
                portName = defaultPortName;
            }
            return portName;
        }

        public static int SetPortBaudRate(int defaultPortBaudRate)
        {
            string baudRate;
            baudRate = "9600";

            if (baudRate == "")
            {
                baudRate = defaultPortBaudRate.ToString();
            }

            return int.Parse(baudRate);
        }

        public static Parity SetPortParity(Parity defaultPortParity)
        {
            string parity;


            parity = "None";

            if (parity == "")
            {
                parity = defaultPortParity.ToString();
            }

            return (Parity)Enum.Parse(typeof(Parity), parity);
        }

        public static int SetPortDataBits(int defaultPortDataBits)
        {
            string dataBits;

            //Console.Write("Data Bits({0}): ", defaultPortDataBits);
            dataBits = "8";// Console.ReadLine();

            if (dataBits == "")
            {
                dataBits = defaultPortDataBits.ToString();
            }

            return int.Parse(dataBits);
        }

        public static StopBits SetPortStopBits(StopBits defaultPortStopBits)
        {
            string stopBits;


            stopBits = "None"; //Console.ReadLine();

            if (stopBits == "")
            {
                stopBits = defaultPortStopBits.ToString();
```

```
            }

            return (StopBits)Enum.Parse(typeof(StopBits), stopBits);
        }

        public static Handshake SetPortHandshake(Handshake defaultPortHandshake)
        {
            string handshake;


            handshake = "None";  //Console.ReadLine();

            if (handshake == "")
            {
                handshake = defaultPortHandshake.ToString();
            }

            return (Handshake)Enum.Parse(typeof(Handshake), handshake);
        }

    }
}
```

Classe Count Bloco

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Projeto_Final
{
    public static class CountBloco //classe estática utilizada para a istanciação dos métodos
estáticos
    {
        public static int nuBloco=0;//variável que armazena o numero do bloco de comunicação
        public static double dist = 0; //variável que armazena a distância percorrida
        public static double litros = 0;//váriável que armazena o combustível gasto

    public static void setCount(int count) {     //método que incrementa o numero do bloco
        nuBloco +=count;
        if (nuBloco > 0xff) { nuBloco = 0; }

    }
    public static int  getCount(){              //método que retorna o numero do bloco

        nuBloco++;
        if (nuBloco > 0xff) { nuBloco = 0; }
        return nuBloco;
    }
    public static double distancia(int v) {

        dist = dist + v * 0.00083; //valor constante dessa formula é calculado com base no
tempo de atualização das informações pelo software
        return (dist);
    }
    public static double consumo(double tInj) {
        double consumo = 0;
        litros+=(tInj*1.8)/1000;
        if (litros != 0)
        {
            consumo = dist / litros;

            return consumo;
        }
        else {
            return 0;
        }

    }

  }
}
```

Classe TrataDados

```
using System;
using System.Collections.Generic;
//using System.Linq;
```

```
using System.Text;
using System.IO;


namespace Projeto_Final
{
    class TrataDados
    {


        //Calcula a rotação do motor do veíulo
        public int Rpm(int a, int b) {
            double rpm = 0.0;
            rpm = 0.2 * a * b;
            return ((int)rpm);
        }
        //Calcula a temperatura do líquido de arrefecimento do motor
        public int TempMotor(int a, int b) {
            double temp=0.0;
            temp = a * (b - 100) * 0.1;
            return((int)temp);
        }
        //Calcula a tensão da bateria
        public double TensaoBat(int a, int b){
            double v=0.0;
            v=0.001*a*b;
            return(v);
        }
        //Calcula o valor da relação da sonda lambda
        public double Lambda(int a, int b) {
            double lambda = 0.0;
            lambda = 0.0001 * a * (b - 128) + 1;
            return (lambda);
        }
        //Calcula a velocidade do veículo
        public int Velocidade(int a, int b){
            double vel=0.0;
            vel=0.01*a*b;
            return((int)vel);
        }
        //Calcula o tempo de injeção do veículo
        public int TempoInjecao(int a, int b) {
           double tempInj = 0.0;
           tempInj = a * b * 0.01;
           return((int)tempInj);
        }

        //Calcula consumo de combustivel
        public double consumo() {
            double cons=0.0;
            return (cons);
        }

        //busca a descrição do erro em um arquivo texto de acordo com com os bytes recebidos
        public string erroUce(int a , int b){
            string comp = "";
            string err = "Erro não catalogado";
            int erro = 0;
            erro=Convert.ToInt32(a.ToString("X") + b.ToString("X"), 16);
            StreamReader sr = new StreamReader("codigos de erro.txt");
            while (!sr.EndOfStream) {
                comp=sr.ReadLine();
                if (comp== erro.ToString("D5")) {
                    err = sr.ReadLine();
                }
            }
            sr.Close();
           return (erro+":"+err);      //retorna numero e descrição do erro
        }

    }
}
```

Classe Comunicacao

```
using System;
```

```
using System.Collections.Generic;
using System.Text;
using System.IO;
using System.IO.Ports;

namespace Projeto_Final
{
    class Comunicacao
    {
       public static void Main()
        {

            Comunicacao comm=new Comunicacao();         //instância da classe Comunicação,
usada para acessar metodos no formulário
            int addEcu;                                 //Endereço de Inicialização da ECU, o
KW1
            int[,] idEcu=new int[5,12];                 //array que armazena os dados de
identificação da UCE
            int[] dados = new int[16];                  //array que armazena os dados de bordo
do veículo
            int[] erro = new int[16];                   //array que armazena os erros
provenientes da UCE do veículo
            Conexao con = new Conexao();                //instancia da classe de conexão
            int rpm=0;                                  //RPM - rotações por minuto
            double temp=0;                              //temperatura do líquido de
arrefecimento
            double tensaoBat=0;                         //Tensão da bateria
            double TInj = 0;                            //tempo de injeção
            double lambda=0;                            //relação estequiométrica
            double consumo = 0;                         // variável que armazena o consumo de
combustível
            int ctrlBloco = 0;                          //variável de controle para apresentar
a informação lambda
            int vel = 0;                                //velocidade
            string erroUce="";                          //string que coleta a descrição do
erro da UCE
            TrataDados trtds = new TrataDados();        //instância da classe trataDados
            SerialPort bt = new SerialPort();           //instância de cinexão serial para
transmissão via bluetooth
            bt.PortName = "COM8";                       //configura o numero da porta com
aberta para comunicação bluetooth
            bt.BaudRate = 9600;                         //configura a taxa de sinalização da
porta
            bt.Open();                                  //abre a porta

           //--------------------------------Cabeçalho da aplicação--------------------------
------------------------------

            System.Console.WriteLine("-------------------------------------------------------
----------------------");
            System.Console.WriteLine("|                        UniCEUB - Projeto Final
|");
            System.Console.WriteLine("|                    Sistema de Monitoramento Veicular
|");
            System.Console.WriteLine("|             Alvaro Santana dos Santos Junior - Ra:
2051593/0               |");
            System.Console.WriteLine("|                Interface de Comunicação Serial-
Bluetooth                    |");
            System.Console.WriteLine("-------------------------------------------------------
----------------------");

           //---------------------------------Estabelece conexões com notebook e dispositivo
móvel-----------------------------------------------------

            con.configuraPorta(con);                    //método que efetua a configuração da
porta serial para comunicação com a UCE do veículo
            con.Open();                                 // estabelece conexão com a porta
COM1;
            if (con.IsOpen == true)                     //Verifica se a porta está aberta
            {
                System.Console.WriteLine("");
                System.Console.WriteLine("Porta Serial " + con.PortName + " Inicializada.");
                System.Console.WriteLine("");
            }
            else {
                System.Console.WriteLine("Erro!!! - Porta serial não encontrada. Verifique a
conexão.");
            }
            System.Console.Write("Sincronizando comunicação com dispoitivo móvel: ");
            if (comm.sincroniza(bt) == 1) //estabelece a sincronização com o dispositivo móvel
            {
```

```
                System.Console.WriteLine("Comunicação estabelecida!");
                System.Console.WriteLine(" ");
            }


            //---------------------------------------------Estabelece conexão com UCE do
veículo e adquire-----------------------------------
          //-------------------------------------------------------os dados de identificação
da mesma---------------------------------------

            addEcu=con.conectaUce(con);                 //recebe o endereço da UCE
            if (addEcu == 0x01)                         //verifica se a conexão com a ECU foi
efetuada
            {
                System.Console.WriteLine("Conexão estabelecida com a UCE");
                System.Console.WriteLine("");
                System.Console.WriteLine("Endereço da ECU: 0X0"+addEcu);    //retorna o
enderreço da ECU que é determinado pelo KW1
                idEcu = con.idEcu(con);                                     //função que
retorna os dados de identificação da UCE
                System.Console.WriteLine("");
                //Id da UCE
                System.Console.WriteLine(
                                            "ID: " + (char)idEcu[0, 0] + (char)idEcu[0, 1] +
(char)idEcu[0, 2] + (char)idEcu[0, 3]
                                            + (char)idEcu[0, 4] + (char)idEcu[0, 5] +
(char)idEcu[0, 6]+ (char)idEcu[0, 7] + (char)idEcu[0, 8]
                                            + (char)idEcu[0, 9] + (char)idEcu[0, 10] +
(char)idEcu[0, 11]
                                        );
                //Nome da UCE
                System.Console.WriteLine(
                                            "NOME: " + (char)idEcu[1, 0] + (char)idEcu[1, 1] +
(char)idEcu[1, 2] + (char)idEcu[1, 3]
                                            + (char)idEcu[1, 4] + (char)idEcu[1, 5] +
(char)idEcu[1, 6] + (char)idEcu[1, 7] + (char)idEcu[1, 8]
                                            + (char)idEcu[1, 9] + (char)idEcu[1, 10] +
(char)idEcu[1, 11]+ (char)idEcu[1, 12]+ (char)idEcu[2, 0]
                                            + (char)idEcu[2, 1] + (char)idEcu[2, 2] +
(char)idEcu[2, 3] + (char)idEcu[2, 4]
                                            + (char)idEcu[2, 5] + (char)idEcu[2, 6] +
(char)idEcu[2, 7] + (char)idEcu[2, 8]

                                        );
                //Código do software da UCE
                System.Console.WriteLine(
                                            "COD SOFTWARE: " + (char)idEcu[3, 0] +
(char)idEcu[3, 1] + (char)idEcu[3, 2]+ (char)idEcu[3,3]
                                            + (char)idEcu[3, 4] + (char)idEcu[3, 5] +
(char)idEcu[3, 6] + (char)idEcu[3, 7] + (char)idEcu[3, 8]
                                            + (char)idEcu[3, 9] + (char)idEcu[3, 10]+
(char)idEcu[3,11]

                                        );
                //Código da peça
                System.Console.WriteLine(
                                            "COD PEÇCA: " + (char)idEcu[4, 0] + (char)idEcu[4,
1] + (char)idEcu[4, 2] + (char)idEcu[4, 3]
                                            + (char)idEcu[4, 4] + (char)idEcu[4, 5] +
(char)idEcu[4, 6] + (char)idEcu[4, 7] + (char)idEcu[4, 8]
                                            + (char)idEcu[4, 9] + (char)idEcu[4, 10] +
(char)idEcu[4, 11]

                                        );
            }
            else {
                System.Console.WriteLine("Erro!!! - Não foi possível estabelecer conexão com a
UCE do veículo");
                System.Console.WriteLine("--> Verifique se a ignição está ligada ou se a
interface está conectada na tomada de diagnóstico do veículo");
            }

            con.trataLixo(con);                     //função que ignora bytes recebidos após a
identificação da UCE

            //-------------------------Solicita os erros armazenados na memória de erro da
UCE-------------------------------------------------
            if (con.solicitaErro(con) == 1)
            {
                System.Console.WriteLine(" ");
                erro = con.erroUce(con);
```

```
                //se o valor for diferente de 0, existe um erro na UCE
                //teste para o 1° grupo de erro
                if (erro[2] != 0)
                {
                    erroUce = trtds.erroUce(erro[0], erro[1]);
                    System.Console.WriteLine(erroUce);
                    bt.WriteLine("1"+ erroUce);
                    System.Threading.Thread.Sleep(30);
                }
                //teste para o 2° grupo de erro
                if (erro[5] != 0)
                {
                    erroUce = trtds.erroUce(erro[3], erro[4]);
                    System.Console.WriteLine(erroUce);
                    bt.WriteLine("2" + erroUce);
                    System.Threading.Thread.Sleep(30);
                }
                //teste para o 3° grupo de erro
                if (erro[8] != 0)
                {
                    erroUce = trtds.erroUce(erro[6], erro[7]);
                    System.Console.WriteLine(erroUce);
                    bt.WriteLine("3" + erroUce);
                    System.Threading.Thread.Sleep(30);
                }
                //teste para o 4° grupo de erro
                if (erro[11] != 0)
                {
                    erroUce = trtds.erroUce(erro[9], erro[10]);
                    System.Console.WriteLine(erroUce);
                    bt.WriteLine("4" + erroUce);
                    System.Threading.Thread.Sleep(30);
                }


            }

            //------Requisita as informações de condução coletadas da tomada de diagnóstico e
as envia para o dispositivo móvel,--------------
           //-------------------------por meio de uma trasmissão bluetooth utilizando uma
porta COM----------------------------------------

            System.Console.WriteLine("Trasmitindo informações de condução...");
            bt.RtsEnable = true;
            while (true)                          //loop infinito para aquisição dos dados e
envio das informações
            {

                if (con.solicitaDados(con, 1) == 1){ //solicita o 1° bloco de dados de bordo
da UCE

                        dados = con.blocoDados(con);

                        //RPM
                        rpm = trtds.Rpm(dados[1],dados[2]);
                        System.Console.Write("|\r");

                        bt.WriteLine("R" + rpm);
                        System.Threading.Thread.Sleep(30);     //delay necessário para a
correta recepção da informação pelo dispositvo móvel

                        //temperatura do motor
                        temp = trtds.TempMotor(dados[4], dados[5]);
                        System.Console.Write("/\r");
                        bt.WriteLine("T" + temp);
                }
                else{
                  System.Console.Write("erro: bloco1");
                }
                if (ctrlBloco % 2 != 0){
                   if (con.solicitaDados(con, 2) == 1){              //bloco2

                      dados = con.blocoDados(con);

                      //Tempo de Injeção
                      TInj = trtds.TempoInjecao(dados[4],dados[5]);
                      System.Console.Write("-\r");
                      bt.WriteLine("I"+TInj);
                      System.Threading.Thread.Sleep(30);

                      //RPM
```

```
                rpm = trtds.Rpm(dados[1], dados[2]);
                //System.Console.WriteLine(rpm + "RPM ");
                System.Console.Write("\\\r");
                bt.WriteLine("R" + rpm);
                System.Threading.Thread.Sleep(30);

                //Tensão da Bateria
                tensaoBat = trtds.TensaoBat(dados[7], dados[8]);
                System.Console.Write("|\r");
                bt.WriteLine("U" + tensaoBat.ToString("F2"));
                System.Threading.Thread.Sleep(30);

             }else{
                System.Console.Write("erro: bloco2");
              }
    }

        if (con.solicitaDados(con, 4) == 1)                        //bloco 04
        {
            dados = con.blocoDados(con);

            //RPM
            rpm = trtds.Rpm(dados[1], dados[2]);
            System.Console.Write("/\r");
            bt.WriteLine("R" + rpm);
            System.Threading.Thread.Sleep(30);

            //Tempo de Injeção
            TInj = trtds.TempoInjecao(dados[4], dados[5]);
            System.Console.Write("-\r");
            bt.WriteLine("I" + TInj);
            System.Threading.Thread.Sleep(30);

            //consumo
            consumo = CountBloco.consumo(TInj);
            System.Console.Write("\\\r");
            bt.WriteLine("C" + consumo.ToString("F2"));
            System.Threading.Thread.Sleep(30);

            //Velocidade
            vel= trtds.Velocidade(dados[7],dados[8]);
            System.Console.Write("|\r");
            bt.WriteLine("V"+vel);
            System.Threading.Thread.Sleep(30);

            //distância
            CountBloco.distancia(vel);
            System.Console.Write("/\r");
            bt.WriteLine("D" + CountBloco.dist.ToString("F3"));
            System.Threading.Thread.Sleep(25);

        }else{System.Console.Write("erro: bloco 4");}

        if (ctrlBloco % 2 == 0)
        {
            if (con.solicitaDados(con, 9) == 1)                        //bloco 09
            {
                dados = con.blocoDados(con);

                //lambda
                lambda = trtds.Lambda(dados[1], dados[2]);
                bt.WriteLine("L" + lambda);
                System.Console.Write("-\r");
                System.Threading.Thread.Sleep(30);
            }
            else {
                System.Console.Write("erro: bloco 9");
            }
        }

           System.Console.SetCursorPosition(0,23);
           ctrlBloco++;
     }


 }

//método utilizado para sincronização com o dispositivo móvel
public int sincroniza(SerialPort bt)
{
    string sync = "";
```

```
        while (sync != "sync") {
            bt.WriteLine("sync");
            sync = bt.ReadLine();

        }

        return 1;
    }
}}
```

APÊNDICE B – CÓDIGO FONTE DO SOFTWARE EMBARCADO EM DISPOSITIVO MÓVEL – APRESENTAÇÃO DA INFORMAÇÃO AO CONDUTOR

Classe FrmApresentacao

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.IO.Ports;
using System.IO.Compression;

namespace SmartDeviceProject3
{

    public delegate void recebeDados();   //declaração do método delegado
    public partial class FrmApresentacao : Form
    {
        public FrmApresentacao()
        {

            byte[] dado = new byte[1];
            InitializeComponent();
            System.Windows.Forms.TabControl tab = new TabControl();
            System.Threading.Thread atualiza=new
System.Threading.Thread(atualizaInf);
                Comm rs = new Comm();
```

```
                rs.BaudRate = rs.txSinal();
                rs.PortName = rs.con();
                rs.DataBits = rs.bitDados();
                rs.Handshake = rs.handShake();
                rs.Parity = rs.paridade();
                try
                {
                    rs.Open();
                    if (sincroniza(rs) == 1)
                    {
                        atualiza.Start();
                    }

                }
                catch (Exception ) {
                    MessageBox.Show("Erro de sincronização", "Erro", MessageBoxButtons.OK,
MessageBoxIcon.Exclamation, MessageBoxDefaultButton.Button1);
                }
        }
        //método que sincroniza a comunicação com a aplicação embarcada no notebook
        public int sincroniza(SerialPort rs) {

            string sync = "";
            while (sync != "sync") {
                sync=rs.ReadLine();
                rs.WriteLine("sync");

            }

            return 1;
        }
        public void atualizaDado(){

            Comm rs = new Comm();

                rs.PortName = "COM1";
                rs.BaudRate = 9600;
                //              rs.ReadTimeout = 5000;
                rs.ReadTimeout = 30000;
                rs.Open();

                rs.DtrEnable = true;

                string str;
                str = rs.ReadLine();
                if (Convert.ToString(str[0]) == "1") { txtErro1.Text = str.Substring(1,
str.Length - 1); }                  //coleta o 1° erro do bloco de erros
                if (Convert.ToString(str[0]) == "2") { txtErro2.Text = str.Substring(1,
str.Length - 1); }                  //coleta o 2° erro do bloco de erros
```

```
                if (Convert.ToString(str[0]) == "3") { txtErro3.Text = str.Substring(1,
str.Length - 1); }                  //coleta o 3° erro do bloco de erros

                if (Convert.ToString(str[0]) == "4") { txtErro4.Text = str.Substring(1,
str.Length - 1); }                  //coleta o 4° erro do bloco de erros

                if (Convert.ToString(str[0]) == "R") { txtRpm.Text = str.Substring(1,
str.Length - 1) + " RPM"; }           //coleta a rotação do motor

                if (Convert.ToString(str[0]) == "T") { txtTemp.Text = str.Substring(1,
str.Length - 1) + " °C"; }           //coleta a temperatura do líquido de arrefecimento

                if (Convert.ToString(str[0]) == "I") { txtTInj.Text = str.Substring(1,
str.Length - 1) + " ms"; }           //coleta tempo de injeção

                if (Convert.ToString(str[0]) == "U") { txtTensao.Text = str.Substring(1,
str.Length - 1) + " V"; }          //coleta a tesnão da bateria

                if (Convert.ToString(str[0]) == "C") { txtConsumo.Text = str.Substring(1,
str.Length - 1) + " Km/l"; }      //coleta o consumo

                if (Convert.ToString(str[0]) == "V") { txtVelocidade.Text = str.Substring(1,
str.Length - 1) + " Km/h"; }   //coleta a velocidade

                if (Convert.ToString(str[0]) == "D") { txtDist.Text = str.Substring(1,
str.Length - 1) + " Km"; }           //coleta a distância

                if (Convert.ToString(str[0]) == "L") { txtLambda.Text = str.Substring(1,
str.Length - 1); }                  //coleta a relação lambda ( relação estequiométrica)

                rs.Close();

        }

        public void atualizaInf()

        {

            recebeDados da = new recebeDados(atualizaDado);   //instância do método delegado

                //loop infinito para a aquisição dos dados, para que os campos sejam
atualizados

                //é necessário utilizar o método invoke

                while (1 == 1)

                {

                    txtRpm.Invoke(da);

                    txtTemp.Invoke(da);

                    txtTInj.Invoke(da);

                    txtTensao.Invoke(da);

                    txtVelocidade.Invoke(da);

                    txtDist.Invoke(da);

                    txtLambda.Invoke(da);

                    txtErro1.Invoke(da);

                    txtErro2.Invoke(da);

                    txtErro3.Invoke(da);

                    txtErro4.Invoke(da);

                }



        }

        private void menuItem2_Click(object sender, EventArgs e)

        {

            //método que fecha a aplicação

            if (MessageBox.Show("Deseja sair?", "Confirmação",
MessageBoxButtons.YesNo, MessageBoxIcon.Question,
MessageBoxDefaultButton.Button1)==DialogResult.Yes)

            {

                Comm com = new Comm();
```

```
                com.Close();
                Application.Exit();
            }
        }
```

Classe Comm

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.Text;
using System.IO;
using System.IO.Ports;

namespace SmartDeviceProject3
{
    class Comm : System.IO.Ports.SerialPort
    {
        public const int InfiniteTimeout = 0;

        public string con(){
            string  Nporta="COM1";                              //Nome  Porta  virtual  serial  de
comunicação
            return Nporta;
        }
        public int txSinal(){
            int taxa = 9600;                          //taxa de sinalização da porta seria, 9600
baud
           return (taxa);
        }
        public Parity paridade() {
            string par = "None";                    //Configuração do bit de paridade
            return (Parity)Enum.Parse(typeof(Parity), par,true);
        }
        public int  stopBit() {
            int stop = 1;                           //Configuração do bit de stop
            return (stop);
        }
        public Handshake handShake() {
            string hndSke = "None";                 //Confiuraçã do handshake
            return (Handshake)Enum.Parse(typeof(Handshake),hndSke,true);
        }
        public int bitDados() {
            int bit = 8;
            return(bit);
        }
```

ANEXO A – TABELA ASCII

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	160	A0	á	192	C0	└	224	E0	α
129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
131	83	â	163	A3	ú	195	C3	├	227	E3	π
132	84	ä	164	A4	ñ	196	C4	─	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	═	237	ED	ø
142	8E	Ä	174	AE	«	206	CE	╬	238	EE	ε
143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
144	90	É	176	B0	░	208	D0	╨	240	F0	≡
145	91	æ	177	B1	▒	209	D1	╤	241	F1	±
146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
155	9B	¢	187	BB	╗	219	DB	█	251	FB	√
156	9C	£	188	BC	╝	220	DC	▄	252	FC	ⁿ
157	9D	¥	189	BD	╜	221	DD	▌	253	FD	²
158	9E	₧	190	BE	╛	222	DE	▐	254	FE	■
159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	