MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO PARANÁ

PEDRO BOSQUIERO DA SILVA

SISTEMA DE AUXILIO À DETECÇÃO DE BURLA EM SISTEMAS DE PÓS-TRATAMENTO DE VEÍCULOS COMERCIAIS

CURITIBA

2018

PEDRO BOSQUIERO DA SILVA

SISTEMA DE AUXILIO À DETECÇÃO DE BURLA EM SISTEMAS DE PÓS-TRATAMENTO DE VEÍCULOS COMERCIAIS

Trabalho de conclusão de curso apresentado ao Curso de Engenharia Elétrica com Ênfase em Sistemas Eletrônicos Embarcados da Universidade Federal do Paran como requisito à obteno do titulo do grau de bacharelado em Engenharia Eletrica.
Orientador: Prof. Dr. Henri Frederico Ebers-pacher

CURITIBA

2018

TERMO DE APROVAÇÃO

PEDRO BOSQUIERO DA SILVA

SISTEMA DE AUXILIO À DETECÇÃO DE BURLA EM SISTEMAS DE PÓS-TRATAMENTO DE VEÍCULOS COMERCIAIS

Trabalho de conclusão de curso apresentado como requisito parcial à obtenção do grau de Engenheiro Eletricista com Ênfase em Sistemas Eletrônicos Embarcados, do Setor de Tecnologia da Universidade Federal do Paraná, pela seguinte banca examinadora:

Orientador: Prof. Dr. Henri Frederico Eberspacher
Departamento de Engenharia Elétrica, UFPR

Prof. Dr. Luis Henrique Assumpção Lolis
Departamento de Engenharia Elétrica, UFPR

Prof. M. Waldomiro Soares Yuan
Departamento de Engenharia Elétrica, UFPR

Curitiba, 4 de dezembro de 2018

DEDICATÓRIA

Dedico este trabalho aos meus familiares, meus professores, em especial ao Dr. Henri Frederico Eberspacher que está me orientando no desenvolvimento deste projeto.

AGRADECIMENTOS

Em primeiro lugar agradeço aos meus familiares que sempre me apoiaram em minhas decisões, sendo sempre a fonte de esperança para seguir acreditando.

Agradeço aos meus colegas de turma e amigos que sempre me apoiaram e incentivaram nos momentos difíceis.

Por fim, agradeço à todos os professores que me passaram seus ensinamentos durante todos os anos de estudo, permitindo que fosse possível obter a base necessária para chegar até aqui.

RESUMO

Nos últimos tempos cada vez mais se discute sobre internet das coisas, ou como é comumente conhecido pelo mundo afora, *Internet of Things* (*IoT*). A cada ano que passa cresce o número de aplicações que se utilizam dessa ferramenta para facilitar, modernizar e integrar processos e aplicações visando maior praticidade e comodidade na realização de tarefas e processos presentes em diversas áreas de atuação, seja nas tarefas mais banais, como por exemplo ter a praticidade de controlar elementos de sua casa através do celular, ou até aplicações mais complexas envolvendo redes de comunicação entre veículos. Esse projeto visa propor meios mais confiáveis e práticos para os policiais rodoviários federais executarem suas fiscalizações e conseguirem obter maior êxito em detectar possíveis burlas no sistema de pós-tratamento. No momento, a fiscalização é feita de uma maneira muito precária e pouco confiável. Do ponto de vista de oportunidade tecnológica essa área é muito promissora, visto que não há soluções práticas e baratas na área, portanto com isso em mente o projeto visa o desenvolvimento de um aplicativo, o qual será responsável por analisar as informações obtidas do veículo e fornecer uma diagnose cofiável a respeito do comportamento do veículo em relação ao sistema de controle de emissões, para que os policiais tenham controle maior sobre a ocorrência ou não de burla. Para tal, deverão ser utilizadas técnicas relacionadas a programação, desenvolvimento de aplicativos, conhecimentos na área automobilística, instrumentação e os conhecimentos diversos adquiridos ao longo do curso.

Palavras-chave: Aplicativo. *IoT*. Tecnologia. Caminhões. ARLA32. Proconve. *Blueto-oth*. Burla. Barramento. Android. Instrumentação

ABSTRACT

In the last years it have become even more usual to discuss about *Internet of Things* (*IoT*). At every passing year grows the number of applications that use this tool to make it easier, improve and integrate processes and applications aiming for more practicality and comfort in performing tasks and processes within various areas of expertise, be in normal tasks, like for instance, comfort and practicality when controlling home objects through your cell phone, or even more complex applications involving communication network between vehicles. This project aim to provide more reliable means to the federal highway policeman fulfill it's inspections and obtain a lot more success in keep track of any changes on the the post treatment system. At the moment, the inspections are done in a precarious way, and are almost unreliable. In a technological opportunity point of view, this area is very promissing, therefore with this in mind, the project aim the development of an app, that it's responsible for analyze the information's obtained thorough the vehicle, and provide a reliable diagnostics in regard of the vehicle behavior on the emission control system, for the policeman to have a bigger control about the occurrence or not of system changes. For such, techniques related to programming should be used, as well as knowledge about app development, automotive area, instrumentation and the diverse knowledge acquired throughout the course.

Key-Words: APP. IoT. Technology. Trucks. AdBlue. Proconve. Bluetooth. System. Bus. Android.

LISTA DE FIGURAS

2.1 Scanner de Veículos. 21
2.2 Arquitetura Veicular ECUs 23
2.3 ECU Diesel 25
2.4 Placa da ECU 27
2.5 Sistema EGR 29
2.6 Sistema SCR 29
2.7 Sistema SCR 30
2.8 Pilares POO 32
2.9 Aplicativos 37
2.10 Rede Bluetooth 38
2.11 Interface Android Studio 40
2.12 Dongle Simma Software 42
2.13 Arquitetura Padrão ISO 43
2.14 Estrutura CAN de 11 bits 45
2.15 Estrutura CAN de 29 bits 46
2.16 Típica Rede J1939 em Veículos 50
2.17 Estrutura do Identificador 51
2.18 Conector OBD-II 54
2.19 Conectores ECU 58
2.20 Trennadapter 59

3.1 Arquitetura Modular do Sistema 60
3.2 Arquitetura Funcional do Sistema 62
3.3 Cabo do Sistema 64
3.4 Dongle VNA-BT 65
3.5 Funcionalidades do *Smartphone* no Sistema 68

4.1 Diagrama da classe *MainActivity*. 70
4.2 Diagrama da classe *VeriffActivity*. 72
4.3 Diagrama da classe *ChooseActivity*. 75
4.4 Diagrama da classe *ShowActivity*. 76
4.5 Diagrama da classe *InformationActivity*. 78
4.6 Diagrama da classe *DeviceListActivity*. 80
4.7 Diagrama da classe *ImageActivity*. 81
4.8 Diagrama da classe *Manifest*. 83
4.9 Diagrama da classe *SplashActivity*. 84
4.10 Diagrama da classe *CallActivity*. 85
4.11 Diagrama da classe *UpdateActivity*. 86

5.1 Lista dos Testes Estabelecidos 88
5.2 Teste *Bluetooth* TB1 90
5.3 Teste *Bluetooth* TB2 90
5.4 Seleção de modelo, Teste TP1 92
5.5 Teste Parâmetros TP1 92
5.6 Teste Parâmetros TP2 Pt.I 93

5.7 Teste Parâmetros TP2 Pt. II 93
5.8 Tabela Verdade, Testes TA1 - TA6 95
5.9 Alteração Nível do ARLA32 96
5.10 Alteração Nível de emissões 97
5.11 Alteração Falha de Comunicação e Elétrica 98
5.12 Teste Anormalidade de Parâmetros 99
5.13 Teste Relatório TR1 100
5.14 Exemplo Relatório 101

6.1 Conclusão dos Objetivos Específicos 102

A.1 Cronograma 110

B.1 Tela Inicial e do Call Center 111
B.2 Tela de Atualização e de Seleção de Parâmetros 111
B.3 Tela de Análise e de Relatório 112

C.1 Call Center e Treinamento 113
C.2 Tela Inicial e Informações Gerais 113
C.3 Escolha de Prâmetros e Relatório 114
C.4 Carregamento e Atualização 114
C.5 Exibição de Parâmetros 115

LISTA DE SIGLAS

IoT	*Internet of things*
OBD	*On Board Diagnostic*
AEA	Associação Brasileira de Engenharia Automotiva
SAE	*Society of Automotive Engineers*
PRF	Policia Rodoviária Federal
Proconve	Programa de Controle de Poluição do Ar por Veículos Automotores
Conama	Conselho Nacional do Meio Ambiente
NOx	Óxidos de Nitrogênio
MP	Material Particulado
EGR	*Exhaust Gas Recirculation*
SCR	*Selective Catalytic Reduction*
PMS	Ponto Morto Superior
PMI	Ponto Morto inferior
Arla	Agente Redutor de Líquido Automotivo
CO	Monóxido de Carbono
HC	Hexacloretano
CO2	Dióxido de Carbono
H2O	Monóxido de Dihidrogênio
DPF	*Diesel Particulate Filter*
SDK	*Software Development Kit*
JRE	*Java Runtime Environment*
JDK	*Java Development Kit*
OSGi	*Open Service Gateway Initiative*
CORBA	*Common Object Request Broker Architecture*
PDA	*Personal Digital Assistant*
LAN	*Local Area Network*
WAN	*Wide Area Network*
PAN	*Personal Area Network*
MIL	*Malfunction Indicator Lamp*
VIN	*Vehicle Identification Number*
DTC	*Diagnostic Trouble Code*
IBM	*International Business Machines*
ISO	*International Standardization Organization*
OSI	*Open System Interconnection*
CiA	*CAN in Automation*
SOF	*Start of Frame*
RTR	*Remote Transmission Request*
IDE	*Identifier Extension*
DLC	*Data Length Code*
CRC	*Cyclic Redundancy Check*
EOF	*End of Frame*
IF	*Interframe Space*
SRR	*Substitute Remote Request*

SUMÁRIO

RESUMO 6

ABSTRACT 7

LISTA DE ILUSTRAÇÕES 9

1 INTRODUÇÃO 14
1.1 Contexto . . . 15
1.2 Problema . . . 16
1.3 Motivação . . . 17
1.4 Justificativa . . . 17
1.5 Objetivo Geral . . . 18
1.6 Objetivos Específicos . . . 18
1.7 Estrutura do Documento . . . 18

2 FUNDAMENTAÇÃO TEÓRICA 20
2.1 Estado da Arte . . . 20
2.2 Arquitetura Veicular . . . 22
2.3 ECU . . . 23
2.3.1 Entrada . . . 26
2.3.2 Processamento . . . 26
2.3.3 Saída . . . 26
2.3.4 Gerenciamento de Alimentação . . . 27
2.4 Legislação Proconve P7 . . . 28
2.5 Pós Tratamento dos Gases . . . 30
2.6 Arla 32 . . . 31
2.7 Programação Orientada a Objeto . . . 31
2.7.1 Encapsulamento . . . 33
2.7.2 Herança . . . 33
2.7.3 Polimorfismo . . . 34
2.7.4 Abstração . . . 34
2.8 Java . . . 35
2.9 Aplicativos . . . 36
2.10 Bluetooth . . . 38
2.11 Android Studio . . . 39
2.12 Dongles . . . 41
2.12.1 Dongle Simma Software . . . 42
2.13 Protocolo CAN . . . 42
2.13.1 CAN Normal e CAN Estendida . . . 44
2.13.2 CAN Normal . . . 45
2.13.3 CAN Estendida . . . 46
2.13.4 Tipos de Mensagens . . . 46
2.13.4.1 Pacote de Dados . . . 47
2.13.4.2 Pacote Remoto . . . 47

2.13.4.3 Pacote de Erros 47
2.13.4.4 Pacote de Sobrecarga 47
2.13.4.5 Pacote Válido 48
2.13.5 Checagem de Erros e Confinamento de Falhas 48
2.14 Protocolo J1939 49
2.14.1 ISO11783 50
2.14.2 NMEA2000 50
2.14.3 ISO11992 50
2.14.4 FMS 51
2.14.5 Grupo de Parâmetros 51
2.14.6 Interpretação do Identificador CAN 51
2.14.7 Número do Grupo de Parâmetros 51
2.14.8 Número de Parâmetro Suspeito 52
2.14.9 Grupo de Parâmetros Especiais 52
2.14.9.1 Grupo de Requisição de Parâmetros 52
2.14.9.2 Grupo de Reconhecimento de Prâmetros 53
2.14.9.3 Grupo de Parâmetros de Solicitação de Endereço . . . 53
2.14.9.4 Grupo de Parâmetros de Protocolo de Transporte . . . 53
2.15 Protocolo OBD-II 53
2.16 Canalyzer 56
2.17 Trennadapter 57

3 PROJETO **60**
3.1 Arquitetura Geral 60
3.2 Funcionalidades 62
3.3 Recursos 63
3.3.1 Recursos de Hardware 63
3.3.2 Recursos de Software 65

4 IMPLEMENTAÇÃO **69**
4.1 Layout 69
4.1.1 Tela Principal 69
4.1.2 Tela de Informações Gerais 71
4.1.3 Tela de Escolha de Parâmetros 74
4.1.4 Tela de Análise de Parâmetros 75
4.1.5 Tela de Relatório 77
4.1.6 Outras telas 79
4.2 Lógica de Programação 82
4.3 Lógica - Manifesto 82
4.4 Lógica - Tela de Carregamento 83
4.5 Lógica - Tela de Call Center 84
4.6 Lógica - Tela de Treinamento 85
4.7 Lógica - Tela de Atualização 85
4.8 Pesquisas sobre protocolos usados 86

5 TESTES **87**
5.1 Aplicação Dummy 88
5.2 Conexão Bluetooth 89

5.3 Leitura de Parâmetros 91
5.4 Condições de Anormalidade 93
5.5 Relatório 100

6 CONCLUSÃO 102
6.1 Trabalhos Futuros 105

A CRONOGRAMA 110

B IDEALIZAÇÃO DO LAYOUT 111

C LAYOUTS FINAIS 113

D MANIFESTO 116

E CLASSE - CALLACTIVITY 119

F CLASSE - CHOOSEACTIVITY 122

G CLASSE - DEVICELISTACTIVITY 126

H CLASSE - IMAGEACTIVITY 133

I CLASSE - INFORMATIONACTIVITY 137

J CLASSE - MAINACTIVITY 149

K CLASSE - MAINACTIVITYFRAGMENT 159

L CLASSE - SHOWACTIVITY 161

M CLASSE - SPLASHACTIVITY 173

N CLASSE - TRAININGACTIVITY 176

O CLASSE - UPDATEACTIVITY 178

P CLASSE - VERIFFACTIVITY 180

CAPÍTULO 1

INTRODUÇÃO

No Brasil 60% de tudo o que o país produz e consome é transportado por caminhões nas rodovias (QUEIROLO, 2018). Essa é uma quantia exorbitante de veículos que circulam diariamente em nossas estradas, e sem eles, o país praticamente para, pois o abastecimento da maioria de nossos estabelecimentos depende das viagens e entregas destes caminhões. Uma grande prova disso foi o impacto causado pela greve dos caminhoneiros ocorrida em 2018.

Visto que esse número é bem representativo, a emissão decorrente da queima de diesel é um grande problema, e se todos esses veículos expelissem esses gases sem controle nenhum, tendo em vista a grande concentração de particulados e gases tóxicos, tanto o meio ambiente quanto a saúde humana encontrariam-se em risco.

Os gases resultantes da queima de diesel em caminhões são um dos fatores mais agravantes para a poluição do ar, principalmente por conter altas concentrações de óxidos de nitrogênio (NOx), com base nisso foram desenvolvidos sistemas integrados ao veículo para o tratamento desses gases. Esses sistemas buscam seguir a legislação de emissões do Brasil, chamada de Proconve P7, que exprime a necessidade de um compartimento extra no sistema de pós tratamento dos gases e um tanque de ARLA32. Porém, cada vez mais vê-se casos em que há uma burla nesse sistema onde coloca-se emuladores para enganar o caminhão e fazê-lo acreditar que tudo está funcionando de acordo com o esperado quando na verdade não há borrifamento de ARLA32 em sua devida parte dentro desse sistema. Um grande exemplo disso é o fato do consumo do Arla 32 estar menor do que deveria, e isso é decorrente de fraudes que prejudicam os veículos e também o meio ambiente. A estimativa da Afeevas (Associação dos Fabricantes de Equipamentos para Controle de Emissores Veiculares da América do Sul) é que o deficit chegue a 45%. O cálculo considera o consumo de diesel S-10 no

último trimestre de 2016 e a frota em circulação que exige a adição da substância para garantir eficiência e reduzir a poluição (PIANEGONDA, 2017).

Com a facilidade encontrada nos dias de hoje em acessar a internet em praticamente qualquer lugar, é interessante que os dispositivos também se conectem na internet, isso é um dos conceitos de *IoT*. Esse sistema pode conectar dispositivos, ou ”coisas”, a computadores e *smartphones* juntos e fornecer uma interface para que os usuários interajam (WENBO; QUANYU; ZHENWEI, 2015). ”Até 2020, existirão 34 bilhões de dispositivos computadores, *smartphones*, relógios inteligentes e internet das coisas, em todo o mundo, conectados à internet”(SILVA, 2016). Portanto, percebe-se que *IoT* é uma tendência e por isso será abordada ao longo deste projeto.

Contudo esse projeto visa desenvolver um sistema de diagnose integrado através de *Bluetooth* e do barramento do veículo para poder analisar as mensagens que trafegam no momento, podendo ser utilizado com facilidade por pessoas praticamente leigas, principalmente nas fiscalizações executadas nas rodovias. Fiscalizações essas, feitas com o intuito de verificar dentre outras coisas, a conformidade do sistema de pós tratamento de gases com a lei vigente.

1.1 Contexto

Um dos grandes eventos dos quais a Bosch costuma participar é o fórum da AEA (Associação Brasileira de Engenharia Automotiva), neste fórum são realizadas diversas apresentações sobre os assuntos atuais mais relevantes em relação ao setor automotivo. Uma destas apresentações foi realizada pela policia rodoviária federal, e nela eles exprimiam o problema cada vez mais frequente que estão tendo com os casos de burla no sistema de pós-tratamento dos veículos comerciais e da incapacidade de detectar estes casos dada a maneira atual como as fiscalizações são efetuadas.

Após a apresentação viu-se a oportunidade de desenvolver uma solução para o problema, então uma conversa com os próprios policiais foi realizada em prol do entendimento mais aprofundado do cenário e também das necessidades envolvidas.

Pouco tempo depois, a própria PRF (Policia Rodoviária Federal) entrou em contato com o setor de *Powertrain Systems* da Bosch para agendar uma reunião com uma equipe técnica de engenheiros e discutir sobre uma ideia de solução para o problema deles e também dos requisitos deste projeto.

Na reunião foi decidido que a solução envolveria um sistema de comunicação com o caminhão em conjunto com um aplicativo desenvolvido pelo setor que, segundo os requisitos do próprio cliente, deveria contemplar *features* como: campo para inserção da placa do veículo, interface amigável, possibilidade de tirar até 6 fotos, etc.

Então após a reunião e com base em toda a discussão e *user experience* realizada, a ideia e aplicação do sistema proposto neste TCC foi elaborada, e também dada toda a experiência adquirida com o decorrer do estágio e do curso está sendo possível o desenvolvimento deste sistema.

1.2 Problema

Como o sistema de pós-tratamento dos veículos envolve além do reabastecimento de *diesel*, o reabastecimento também de um tanque de aditivo (ARLA32). Acaba por gerar um custo adicional, pelo qual alguns frotistas e caminhoneiros não estão dispostos a arcar, visto que aparentemente não resulta em uma alteração de desempenho do veículo.

Porém, visto que o aditivo em questão e o próprio sistema de pós-tratamento contribuem de maneira significativa para a diminuição da emissão de poluentes destes veículos e que sua aplicação correta é essencial para o cumprimento da legislação Proconve P7, que regulamenta os níveis de emissões máximos para os sistemas aplicados atualmente, é de suma importância que todos os processos envolvidos sejam respeitados, levando em consideração os males que os particulados e gases NOx causam para o meio ambiente e para a qualidade de vida dos seres humanos.

Porém o grande problema que se enfrenta no momento é a precariedade dos métodos de fiscalização atuais realizados pela PRF em contrapartida com a sofisticação

dos métodos de burla, então a necessidade de um sistema mais confiável, prático e tecnológico é essencial para o cenário.

1.3 Motivação

Após levantar todos os problemas junto à PRF, gerados pela falta de métodos e recursos tecnológicos confiáveis para a realização das fiscalizações e detecção de burla, em conjunto com o aprofundamento dos conhecimentos na área e nos dispositivos e métodos utilizados para realizar estas burlas, surgiu a motivação para a elaboração de um sistema que pudesse fornecer maior praticidade e confiança nas fiscalizações, através da coleta de dados do próprio motor do veículo.

Além disso, outra motivação foi a possibilidade de contribuir com peso para um grande projeto de inovação da empresa e aplicar os conhecimentos adquiridos tanto no estágio quanto no curso, visto que experiência em solucionar problemas é uma das características mais importantes e visadas no perfil de um engenheiro.

1.4 Justificativa

Existe uma norma que regulamenta o nível de emissões a serem controladas pelos veículos aqui no Brasil chamada PROCONVE P7. Teoricamente todos os caminhões deveriam cumprir essas normas para que seus gases de escape poluam a níveis controlados e estipulados por essa norma. Porém como isso envolve um custo maior por parte dos frotistas e caminhoneiros, foram criadas algumas maneiras de burlar esse sistema, seja por alteração de fusíveis ou instalação de emuladores.

Isso é um problema cada vez mais frequente e os métodos atuais que os policiais rodoviários e os agentes do IBAMA usam para tentar detectar essas burlas são bem precários e dificilmente detectam algo de errado. Por isso, faz-se necessária alguma solução mais tecnológica, de fácil uso e que possa dar mais confiança e possibilidade de detectar esses casos de burla. Por isso, a ideia de um sistema que se comunica com o veículo tendo uma interface conhecida, como um aplicativo para mediar essa

interação. Assim visando um cenário onde mais veículos, idealmente todos, passem a cumprir essa legislação e com isso as emissões fiquem realmente menos nocivas para o ambiente e para a saúde das pessoas.

1.5 Objetivo Geral

Desenvolver um sistema que seja capaz de adquirir dados do motor do veículo, por meio do conector de OBD (*On Board Diagnostic*) e utilizando comunicação *Bluetooth* para transmitir e interpretar as mensagens de protocolo SAE (*Society of Automotive Engineers*) J1939 presentes no barramento de dados desses veículos, para que posteriormente possam ser analisadas, classificadas, gerando um relatório, visando prover um embasamento mais confiável para os policiais rodoviários em suas fiscalizações.

1.6 Objetivos Específicos

- Realizar estudos sobre normas e legislações;
- Analisar requisitos e elaborar proposta;
- Projeto do sistema;
- Implementação da interface e da comunicação;
- Desenvolvimento da coleta e da interpretação de dados;
- Codificações finais e testes.

1.7 Estrutura do Documento

Este documento possui em sua estrutura, ao todo, 6 capítulos. O capítulo 2 contemplará o embasamento teórico necessário para compreensão, implementação e desenvolvimento do projeto, tendo seu enfoque nos conteúdos que não foram vistos no decorrer do curso. O capítulo 3 explica sobre quais são os recursos de *software* e *hardware*

envolvidos no projeto e também a justificativa do porque foram escolhidos. O capítulo 4 trata da parte de implementação do projeto, onde todo o embasamento anterior é aplicado para atingir os objetivos. O capítulo 5 contém as descrições de todos os processos de como foram testadas as funcionalidades. E o capítulo 6 apresenta a ideia geral sobre os resultados destes testes, sempre levando em conta e colocando de maneira clara qual o grau de realização dos objetivos específicos.

CAPÍTULO 2

FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão apresentados os conceitos necessários envolvidos para a compreensão do projeto, envolvendo protocolos, normas, sistemas e conceitos acadêmicos.

2.1 Estado da Arte

O Estado da Arte é uma parte importante desse projeto, uma vez que faz referência ao que já se tem sobre dispositivos para análise de mensagens em veículos. Além disso, auxilia no desenvolvimento de novos conceitos e ideias, visando a melhoria do projeto como um todo e seus diferenciais em relação aos equipamentos já existentes.

Atualmente as fiscalizações feitas pelos policiais rodoviários federais em relação à adequação ou não do veículo em questão, a respeito dos níveis de emissões, são bem precárias. Essa fiscalização consiste basicamente de quatro etapas.

- Uma entrevista com o motorista;
- Ligar veículo e verificar luzes dos painéis;
- Verificar se o tanque de ARLA32 (Agente Redutor de Líquido Automotivo) não está vazio;
- Através de uma cartilha com diversas cores de gases de escapamento verificar se a coloração está dentro dos padrões.

No momento não existem soluções tecnológicas viáveis em vigor para o auxílio dessas fiscalizações e também não existe nada que seja voltado especialmente para essa detecção de burla do sistema de pós-tratamento, bem como análise de conformidade de níveis de emissões para com a legislação vigente.

O mais próximo que existe no mercado atual para verificar informações do veículo são os chamados *scanners*, equipamentos eletrônicos que se comunicam com o motor do veículo e conseguem exibir algumas informações, tais como: velocidade do motor, distância total rodada pelo veículo, temperatura do motor, nível de óleo, etc. Porém, esses equipamentos são muito caros e relativamente complicados de serem utilizados por pessoas leigas, visto que necessitam de treinamento do usuário e também do fornecimento de manuais e a cada atualização também há a possibilidade de mudar os comandos e no caso o público alvo seriam os policiais rodoviários federais e agentes do IBAMA que não possuem conhecimento aprofundado sobre esses aparelhos e como são feitas as requisições de dados. Um exemplo de *scanner* pode ser visto na Figura 2.1.

Figura 2.1: Scanner de Veículos.

Fonte: Chiptronic Store (2017).

Foi pensando nessa necessidade de mercado, analisando a falta de soluções baratas e práticas, levando em conta a taxa de burla vem crescendo desenfreadamente e que essas emissões sem tratamento tem grande impacto ambiental e na saúde das pessoas que foi decidido executar o desenvolvimento do sistema proposto neste TCC.

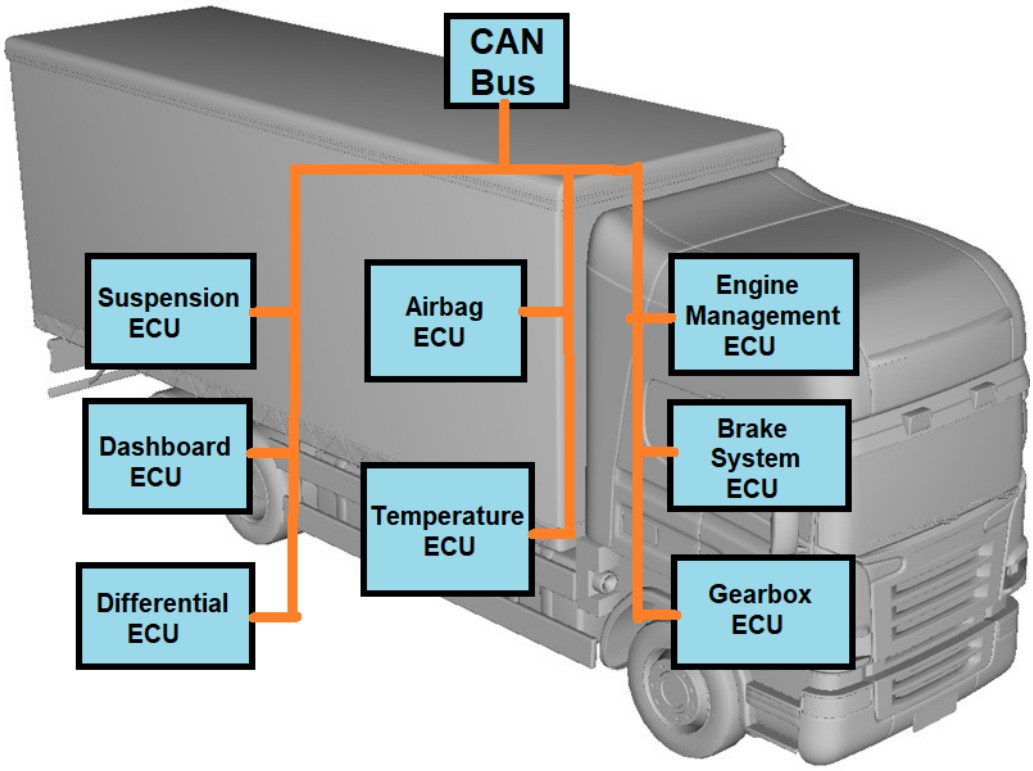
2.2 Arquitetura Veicular

Segurança, dirigibilidade, propulsão e entretenimento são alguns dos muitos sistemas presentes em veiculos que dependem de arquiteturas eletroeletrônicas. Os motoristas de hoje esperam que seus carros e caminhões funcionem sem falhas no período que os possuírem, mas mesmo assim permanecem leigos às complexidades dos sistemas eletrônicos que fazem todos os processos funcionarem. A maioria dos veiculos possuem mais de 1,6 quilômetros de cabos, diversas ECUs (*Electronic Control Unit*), e muitas ligações seriais para implementar as funcionalidades necessárias para o mercado atual (DELPHI, 2007).

Já se foram os dias em que os sistemas eletrônicos veiculares controlavam apenas um ou dois acessórios. Conforme os veiculos foram ficando mais sofisticados em termos funcionalidades eletrônicas, como sistemas de navegação, radio por satelite, airbags, sistemas de controle de temperatura e alimentação, sistemas de pós-tratamento. O desenvolvimento da arquitetura eletroeletrônica veicular passou a ter um papel muito mais importante na industria do transporte. (DELPHI, 2007).

Em um primeiro momento de desenvolvimento desses sistemas eletrônicos, os veiculos possuíam uma ECU que controlava as poucas funções disponíveis na época, portanto a arquitetura era simples e não era necessário a preocupação com cabeamento e problemas complexos de *software*. Porém. com o passar dos anos, com a demanda cada vez maior de funcionalidades, com o aumento de componentes eletrônicos e acompanhando o refinamento e aprimoramento dos sistemas e da própria arquitetura eletroeletrônica, os veiculos passaram a adotar diversas ECUs que passaram a controlar funcionalidades especificas e então passou a ser necessário a preocupação com a arquitetura de como essas ECUs iriam ser instaladas no veiculo, a quantidade, como seriam divididas duas funcionalidades e o principal, por qual método se comunicariam para a integração e funcionamento de todo sistema. Posteriormente essa rede de comunicações entre ECUs passou a ser conhecida como barramentos CAN. Um exemplo da arquitetura veicular moderna pode ser visto na Figura 2.2.

Figura 2.2: Arquitetura Veicular ECUs



Fonte: O Autor (2018).

A eletrônica digital veicular tem sido monitorada, controlada e gerenciada virtualmente há muito tempo. Então não é surpresa que com o aprimoramento e sofisticação dos motores Euro 6, os envolvidos na área precisam focar seus conhecimentos na comunicação por barramentos CAN. Mas além disso, também significa a necessidade de um aprofundamento na aprendizagem sobre os sistemas de cabeamento e também em como funciona a rede de comunicação entre ECUs (ENGINEER, 2014).

2.3 ECU

No indústria automotiva, o termo ECU costuma se referir à Electronic Control Unit, que nada mais são que controladoras de diversas funções do veiculo, como pro exemplo controle de injeção de combustível, tempo para faísca, etc. Um motor de combustão interna é essencialmente uma grande bomba de ar que se movimenta utilizando combustível. Conforme o ar é sugado, é preciso fornecer combustível o bastante para

criar poder o bastante para manter o motor em funcionamento. Essa combinação de ar e combustível é chamada de 'mistura'. Se a mistura for rica demais, o motor vai rodar com muito mais poder que o necessário, acarretando em gastos desnecessários de combustível. Ao passo que se for pobre demais, não terá força o bastante para fazer o caminhão andar (TESTING, 2016).

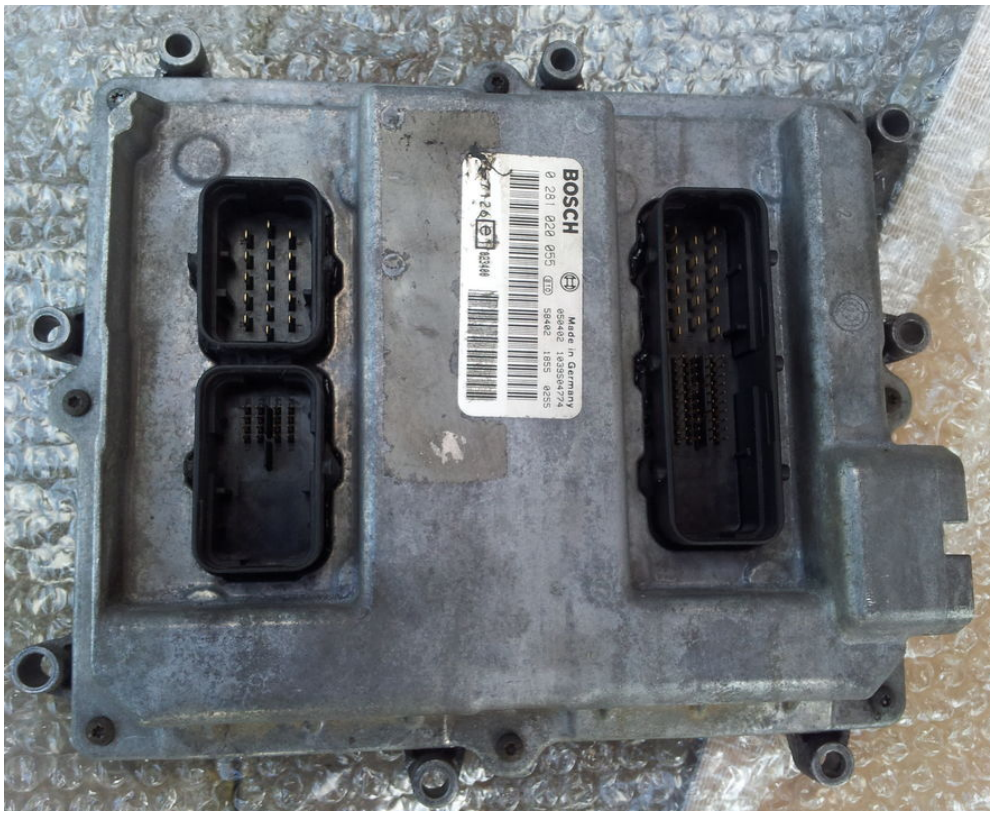
Não só a quantidade de mistura é importante, como também a razão dessa mistura precisa estar correta. Muito combustível e pouco oxigênio faz com que a combustão seja suja e desperdiçada. Pouco combustível e muito oxigênio faz com que a combustão seja lenta e fraca. Nos primórdios, motores costumavam ter essa razão controlada pelo carburador. Mas com as demandas dos veiculos modernos se focando em eficiência energética e baixas emissões, a mistura precisava ser melhor controlada. E a única forma de cumprir esses requisitos rigorosos era passando o controle do motor para uma ECU. Essa ECU tem o dever de controlar a injeção de combustível e ignição utilizando equações pré-estabelecidas e tabelas numéricas.

A ECU também possui a funcionalidade de ligar os componentes de um veículo através dos barramentos CAN, esses componentes do veiculo são interligados, sejam eles memórias, processadores, pinos para conexão com sensores e atuadores, circuitos eletrônicos compostos por elementos passivos e ativos, entre outros. Embora existam diferentes fabricantes de ECUs e cada um com sua arquitetura específica, existem alguns elementos comuns entre elas, tais como:

- Microcontrolador;
- Memória volátil (SRAM);
- Memória não volátil (EEPROM, Flash);
- Valores de entrada de alimentação (tensão), entradas digitais, analógicas e outras;
- Valores de saída como drivers de relés, ponte H, saídas lógicas e outras;
- Periféricos diversos, por exemplo, para comunicação entre ECUs, diagnósticos, etc.

Na Figura 2.3 é possível observar uma típica ECU de caminhões diesel.

Figura 2.3: ECU Diesel



Fonte: Autoline (2017).

Em 1970 as ECUs exerciam a função apenas de controlar alguns solenoides nos carburadores para que eles funcionassem de maneira mais eficiente. Alguns começaram controlando a mistura em velocidades constantes. Já em 1980, com a introdução da injeção de combustível, a ECU assumiu um novo papel de ser completamente responsável pelo gerenciamento do combustível e da ignição em motores à ciclo otto. Pouco tempo depois o controle do lambda em regime fechado também foi incluído e a ECU começou uma nova era de eficiência. Em 1990 a ECU passou a cuidar também da segurança do veiculo. Também foi o início da aparição dos motores Diesel (TESTING, 2016).

Nos anos 2000, a adoção do controle da borboleta por aceleração eletrônica, controle do *turbocharger* e diversos sistemas de emissão foram adicionados às funcionalidade da ECU. No cenário de 2010 para frente, a ECU agora possui controle total

sob a mistura de combustão, abertura da borboleta e dos sistemas de refrigeração e emissões. Ela pode ter mais de 100 entradas e saídas e faz parte de uma rede de diversas outras ECUs dentro do veiculo. Sistemas híbridos se baseiam na comunicação com ECUs para funcionar corretamente, enquanto as funções de auxilio ao motorista se comunicam para tomar o controle da demanda quando necessário (TESTING, 2016).

Uma ECU é comumente conhecida como o 'cérebro' do motor. É essencialmente um computador, um sistema de comunicação e gerenciamento de informações em pequena escala. Para funcionar mesmo em nível básico, é necessário incorporar 4 áreas de operações diferentes.

2.3.1 Entrada

Inclui tipicamente sensores de pressão e temperatura, sinais digitais e dados de outros módulos dentro do veiculo, assim a ECU precisa coletar as informações e decidir quais decisões tomar. Um exemplo de entrada seria a leitura do sensor de temperatura do líquido de refrigeração, ou da leitura do sensor da posição do pedal do acelerador. Solicitações do módulo ABS (Antilock Brake System) podem também ser considerados, para aplicação do controle de tração (TESTING, 2016).

2.3.2 Processamento

Assim que os dados tiverem sido coletados pela ECU, o processador precisa determinar especificações de saída, tais como largura de pulso do injetor de combustível. Essas informações são comandadas pelo *software* imbutido na ECU. O processador não só lê o software para decidir quais saídas são as mais apropriadas, como também grava suas próprias informações, como as misturas ideias e ajustes (TESTING, 2016).

2.3.3 Saída

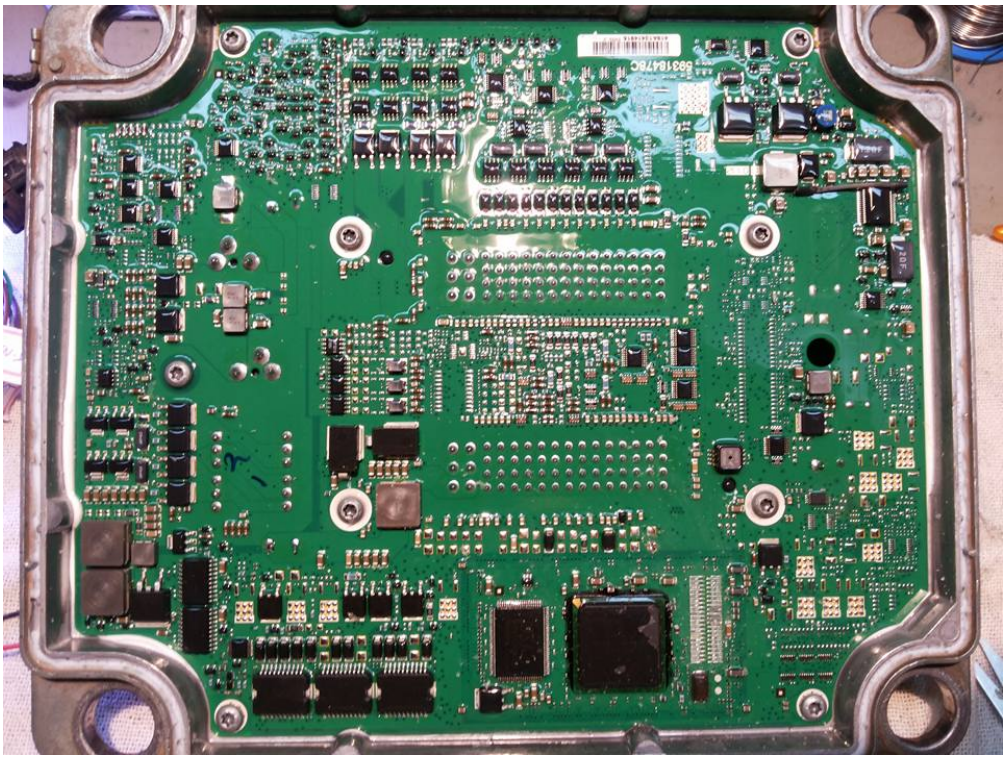
A ECU pode então realizar uma ação no motor, permitindo a correta quantia de alimentação controlar os atuadores com precisão. Isso pode incluir o controle da

largura de pulso da injeção de combustível, o momento exato para realizar a combustão, abertura da válvula eletrônica ou ativação do ventilador de refrigeração (TESTING, 2016).

2.3.4 Gerenciamento de Alimentação

A ECU possui muitos requisitos de alimentação internos para seus milhares de componentes funcionarem de maneira correta. Em adição à isso, para que os diversos sensores e atuadores funcionem, a tensão correta precisa ser fornecida pela ECU para os componentes do veiculo. Pode ser uma simples alimentação de 5 volts para os sensores, ou mais de 200 volts para os circuitos de injeção de combustível, Não apenas a tensão precisa estar correta, mas algumas saídas precisam lidar com mais de 30 amperes, o que naturalmente, gera muito calor. Portanto o gerenciamento térmico é um fator chave no design de uma ECU (TESTING, 2016). Na Figura 2.4 é possível observar como os diversos componentes estão dispostos na placa da ECU.

Figura 2.4: Placa da ECU



Fonte: Nairaland (2016).

2.4 Legislação Proconve P7

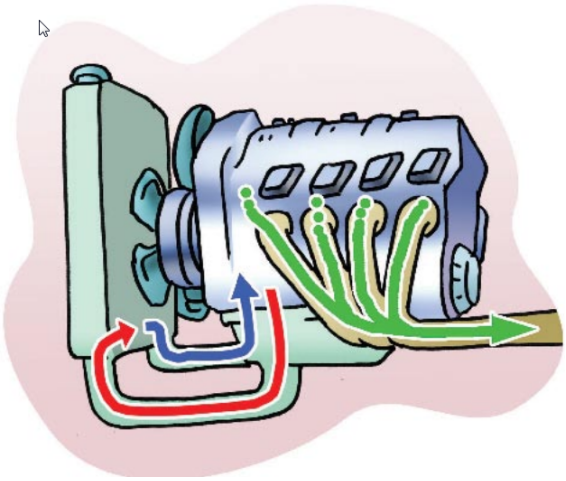
O Proconve (Programa de Controle de Poluição do Ar por Veículos Automotores) foi criado em 1986 pelo Conama (Conselho Nacional do Meio Ambiente) como uma forma de controle de qualidade do ar nos centros urbanos. Ele nada mais é que uma adaptação das metodologias internacionais, especialmente o Euro, às necessidades brasileiras. O Proconve P7 é a versão brasileira para o Euro V, que já está em vigor na Europa e representa a quinta etapa de diminuição progressiva de emissão de gases (IVECO, 2011).

No Brasil, essas medidas de controle de níveis de emissões vêm sendo implantadas gradativamente através do Proconve, visto os problemas que as toxinas decorrentes da queima de combustível causam tanto ao ser humano quanto ao ambiente. Esse programa garante que todos os veículos nacionais e importados que circulam pelo país, tenham seu regime de operação dentro dos limites de emissões permitidos, onde todas as emissões do escapamento são minuciosamente testadas. Em especial, na versão P7, vigente hoje em dia encontra-se a instalação de um tanque com o aditivo ARLA32 para melhor controle e diminuição dessas toxinas expelidas, em especial os óxidos de nitrogênio.

A nova legislação P7 traz redução de 60% de NOx e de 80% das emissões de MP (Material Particulado) em relação às versões anteriores (ANFAVEA, 2012).

Como o problema de emissões de gáses tóxicos foi se tornando algo cada vez mais relevante, viu-se a necessidade de implementar mudanças. Dentre as principais mudanças para atender aos novos limites de emissões está a nova versão do programa que compreende a tecnologia EGR (*Exhaust Gas Recirculation*), que nada mais é que uma recirculação do gás de escapamento, por meio do qual esse gás retorna à admissão, reduzindo a temperatura na hora da combustão e eliminando boa parte do NOx. Esse sistema de recirculação pode ser visto na Figura 2.5.

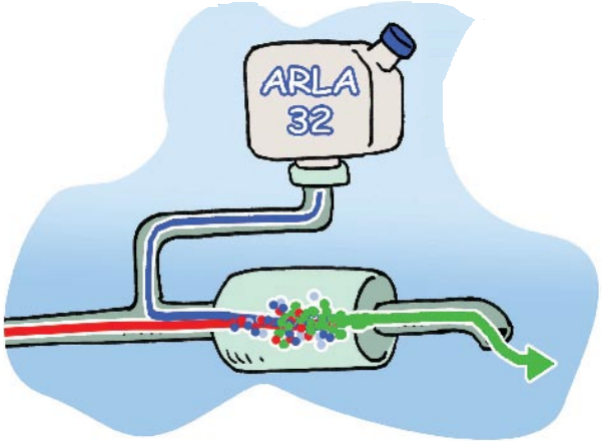
Figura 2.5: Sistema EGR



Fonte: ANFAVEA (2017).

Outra tecnologia é a SCR (*Selective Catalytic Reduction*), onde um aditivo, ARLA32, é pulverizado no gás de escapamento, ocorrendo uma reação química no catalizador que praticamente neutraliza a geração de NOx (ANFAVEA, 2012). Essa é a etapa principal desse sistema para diminuir o nível de emisões. A dosagem é feita por um bico injetor especial formatado especialmente para a pulverização dessa solução. e a calibração do sistema eletrônico dita qual a quantidade que deve ser injetada para obter um resultado que atenda a legislação de emissões. Esse sistema com aditivo pode ser visto na Figura 2.6.

Figura 2.6: Sistema SCR

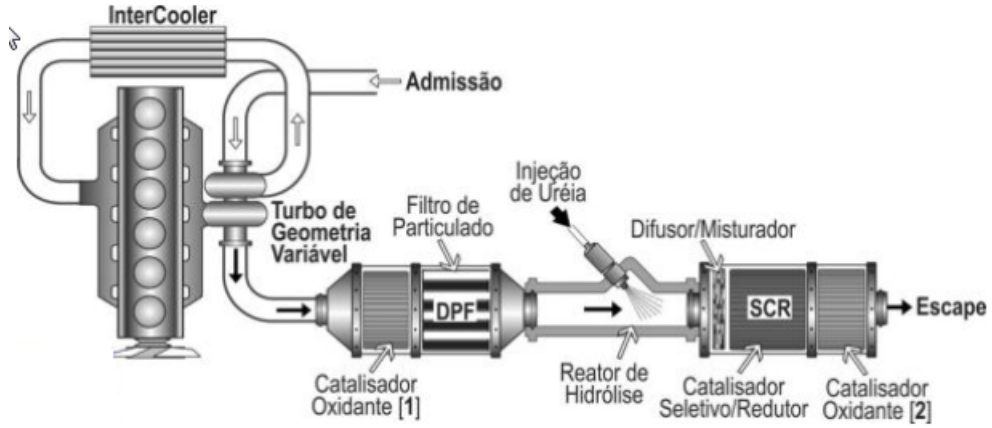


Fonte: ANFAVEA (2017).

2.5 Pós Tratamento dos Gases

Como a combustão no processo de um motor diesel de quatro tempos é rápida, isso propicia o aumento considerável de pressão da câmara e, com isto, picos de alta temperatura, o que resulta na formação de NOx. Por outro lado, a quantidade máxima de combustível injetado deve estar de acordo com a massa de ar admitida de forma a minimizar o particulado emitido (MANAVELLA, 2015). A Figura 2.7 mostra a configuração do sistema de pós-tratamento utilizado pela maioria dos caminhões que possuem sistema SCR.

Figura 2.7: Sistema SCR



Fonte: Oficina Brasil (2017).

Os gases decorrentes da combustão que saem na etapa de descarga possuem em sua composição como principais poluentes os particulados juntamente com NOx. Como esses gases são altamente nocivos para o ambiente e para a população, foi necessário o desenvolvimento de um sistema de pós tratamento para diminuir a concentração desses resíduos tóxicos e assim, tornar as emissões menos nocivas (MANAVELLA, 2015).

Primeiro o catalizador oxidante trata as emissões de CO e HC que resultam em CO2 e H2O. Depois, o DPF (*Diesel Particulate Filter*) retém o material particulado. Quando o acúmulo ultrapassa o limite, o filtro deve ser regenerado através da combustão das partículas. Isso é feito com a pós-injeção de combustível que, ao queimar o catalizador oxidante [1], eleva a temperatura dos gases, o que permite a entrada em combustão

regenerando o filtro (MANAVELLA, 2015). Em seguida, o SCR trata da redução das emissões de NOx. Para seu funcionamento, o SCR utiliza amônia que, ao reagir com o NOx, resulta em N2 e H2O. A amônia (gás tóxico), necessário à reação, é obtido a partir de uma solução de 32% de ureia (substância não tóxica), dosada antes do SCR. Com a alta temperatura dos gases de escape (acima de 170 graus), a ureia é transformada em amoníaco no reator de hidrólise (MANAVELLA, 2015). Por fim, o catalizador oxidante [2] cumpre a função de eliminar qualquer resto de amoníaco não utilizado no SCR, transformando-o em N2 e H2O.

2.6 Arla 32

Como visto no sistema SCR explicado na sessão de pós tratamento dos gases, é utilizado nesse processo um aditivo chamado Arla 32, ele também é conhecido como *AdBlue* na Europa e DEF nos EUA. É um reagente químico a base de ureia de alta pureza projetado especificamente para uso em sistemas SCR com o objetivo de reduzir as emissões de NOx. Ele é incolor, estável e não inflamável. O 32 é adicionado ao nome pela concentração combinada de ureia ser de 32,5% e os outros 67,5% de água deionizada. Todo veículo cujo sistema de pós tratamento tem como base o SCR, possui um tanque específico para esse aditivo. Os veículos que fazem uso do Arla 32 possuem indicadores no painel que mostram para o motorista a quantidade restante do fluido (ALMEIDA, 2013).

2.7 Programação Orientada a Objeto

A primeira característica da Programação Orientada a Objeto, se refere ao método de programação e o modelo da linguagem. Embora tenha muitas interpretações de Orientação a Objeto, uma primeira ideia é modelar um software para que vários tipos de dados manipulados sejam combinados com suas operações relevantes. Então, dados e códigos são combinados em entidades chamadas objetos. Um objeto pode ser imaginado como um pacote que contém comportamento (código) e estado (dado).

O princípio é separar o que muda do que permanece o mesmo, e a programação orientada a objeto possui quatro pilares fundamentais: polimorfismo, herança, abstração e encapsulamento (GUIDE, 2016). É possível observar melhor esses pilares seguindo a Figura 2.8

Figura 2.8: Pilares POO

Fonte: Ateo Momento (2017).

Frequentemente, a mudança de alguma estrutura de dados requer uma mudança correspondente no código que opera os dados ou vice e versa. Essa ideia fornece uma fundamentação mais estável para desenvolver softwares. A intenção é de fazer um grande software ser mais fácil de manusear, assim aumentando a qualidade e reduzindo as falhas. Outro objetivo da Programação Orientada a Objeto é desenvolver objetos mais genéricos para que o software possa tornar o projeto reutilizável. Um objeto genérico chamado “cliente”, por exemplo, deveria ter, a grosso modo, um mesmo comportamento básico em diferentes projetos. Neste sentido, objetos podem ser vistos mais como componentes plugáveis, ajudando a indústria do software a construir grandes projetos a partir de peças existentes e bem montadas, assim guiando a uma massiva redução no tempo de desenvolvimento (GUIDE, 2016).

Relacionando a Programação Orientada a Objeto à linguagem Java, é conhecido que os tipos primitivos do Java não são objetos. Os tipos primitivos contém seus valores na pilha em vez de serem referenciados aos valores. Isto foi pensado pelos desenvolvedores do Java por uma questão de performance. Por isso, o Java não é considerado puramente orientado a objeto. Entretanto, no Java 5.0, com o auxílio do *autoboxing* se

tornou possível programadores escreverem como se os tipos primitivos fossem classes e livremente mudar entre eles para aumentar a flexibilidade. Os desenvolvedores do Java decidiram não implementar certas ferramentas presentes em outras linguagens orientadas a objeto, como: múltiplas heranças, operador sobrecarregado, propriedades de classes, tuplas (GUIDE, 2016).

Como dito anteriormente, há quatro pilares na Programação Orientada a Objeto: Encapsulamento, Herança, Polimorfismo e Abstração. O desenvolvimento dos conceitos foram voltados para a linguagem utilizada no projeto, ou seja, o Java.

2.7.1 Encapsulamento

Encapsulamento significa colocar junto todas as variáveis e métodos em uma única unidade chamada Classe. Assim, pode ser dito também que se trata de esconder os dados e os métodos dentro de um Objeto. Encapsulamento provê a segurança que mantém os dados e métodos seguros de mudanças indesejadas. Programadores as vezes se referem ao encapsulamento como o uso de uma ”caixa preta”, ou um dispositivo com o qual é possível usá-lo sem a necessidade de contato com os mecanismos internos. Um programador pode acessar e usar os métodos e dados contidos dentro desta ”caixa preta”, mas não pode modificá-los, esse método facilita e garante uma abrangência maior de aplicações pois o programador não precisa se preocupar com a interferência indevida de usuários, seja para *hackear* a aplicação, para contornar algum procedimento necessário ou até mesmo para evitar que outros desenvolvedores amadores façam alterações sem conhecimento (W3RESOURCE, 2017).

2.7.2 Herança

A Herança pode ser definida como a habilidade de criar classes que compartilham os atributos e métodos de outras classes existentes, mas com características mais específicas. Herança é geralmente usada para códigos reutilizáveis. Isso é feito para que se possa fazer uso de um código com classes já escritas e ampliar sua utilidade.

Assim, novas classes derivando de classes já existentes, é conhecido como Herança (W3RESOURCE, 2017). Um exemplo que vale ser citado é quando em uma aplicação se tem muitas classes que precisam de métodos similares para serem implementadas e ao invés de ficar reescrevendo para cada uma delas os métodos em comum, cria-se essa classe superior contendo todos esses métodos e aquelas classes inferiores herdam todas as características e podem usá-los de maneira eficiente.

2.7.3 Polimorfismo

O Polimorfismo é definido como: Poli que significa muitas e Morfo que significa forma. Isso descreve a característica de linguagens que permitem a mesma palavra ou símbolo de ser interpretado corretamente em diferentes situações baseadas no contexto. Há dois tipos de Polimorfismo disponíveis em Java. Por exemplo, na língua Portuguesa temos diferentes significados para uma mesma palavra. O entendimento é feito dependendo do contexto. Os programas que utilizam Programação Orientada a Objeto são escritos para que os métodos que tenham o mesmo nome funcionem diferentemente em diferentes contextos. Os tipos de Polimorfismos disponíveis em Java são: Estático e Dinâmico. Polimorfismo Estático: Habilidade de implementar diferentes métodos apenas alterando o argumento usado para chamar o método é conhecido como *overload*. Polimorfismo Dinâmico: Quando uma Classe filha é criada, essa Classe contém dados e métodos que foram definidos na Classe mãe. Em outras palavras, qualquer objeto da Classe filha tem todos os atributos da Classe mãe. As vezes, entretanto, os dados e os métodos da Classe mãe não são totalmente apropriados pelos objetos da classe filha. Nesses casos, é desejável fazer o *override* da classe mãe (W3RESOURCE, 2017).

2.7.4 Abstração

Todas as linguagens de programação apresentam abstração. Pode ser argumentado que a complexidade dos problemas que são possíveis resolver está diretamente relacionado ao tipo e qualidade de abstração. Um elemento essencial da Programação

Orientada a Objeto é a Abstração. Um poderoso meio de tratar abstração é através do uso de classificações hierárquicas. Isto permite que a camada de semântica de sistemas complexos, quebrando em vários pedaços. Por fora, um carro é um simples objeto. Por dentro, o carro consiste de vários subsistemas: direção, freios, cinto de segurança, etc. Cada um desses subsistemas é feito de unidades especializadas. A ideia é tratar a complexidade do carro em várias divisões. Uma Classe abstrata é algo que é incompleto e não pode ser instanciada a partir disto. Se desejável utilizá-lo é preciso fazer com que seja concreto, estendendo-o. Uma Classe é concreta se não contém nenhum método abstrato e implementa todos os métodos abstratos a partir de Herança de uma Classe abstrata ou Interface, através de *implements* ou *extends*. Java tem o conceito de classes abstratas e métodos abstratos, porém uma variável não pode ser abstrata (W3RESOURCE, 2017).

2.8 Java

A linguagem de programação Java começou como um projeto chamado "*Oak*" desenvolvido por James Gosling e seus colegas na Sun Microsystems no começo da década de 90. O objetivo de Gosling era de implementar uma máquina virtual e uma linguagem que fosse parecida com C, mas com maior uniformidade e simplicidade (GUIDE, 2016).

A explicação da origem do nome *Oak* foi que enquanto pensava em uma estrutura de diretórios para a linguagem, observava pela janela um carvalho. Mas esse nome já estava registrado, então o nome acabou surgindo na cafeteria local da cidade onde tomavam café. "*Java*", pois era o nome da terra de origem do café, que os programadores da equipe apreciavam nessa cafeteria, por isso que a logo do Java é um café (VINíCIUS, 2016).

Diferente de outras linguagens as quais são geralmente designadas para serem compiladas com código nativo ou para ser interpretadas da fonte, Java foi feita para ser compilado como *bytecode*, o qual é rodado pela Java Virtual Machine. Por isso, a linguagem Java por si mesma tem síntaxe parecida com C e C++, mas tem um modelo

de objeto mais simples e poucas ferramentas de baixo-nível. O primeiro Java a vir a público foi em 1995, com a promessa de ser escrito apenas uma vez o código e ele poder rodar em qualquer lugar. Tinha uma certa segurança e isto era configurável, permitindo a conexão e os arquivos serem limitados. Novas versões para grandes e pequenas plataformas logo foram designadas com a chegada do “Java 2” (GUIDE, 2016).

O Java permanece uma propriedade controlada pela Java Community Process. A Sun Microsystems faz a maioria das implementações do Java disponíveis sem custos, com sua receita sendo gerada de produtos especializados como o Java Enterprise System. A empresa diferencia entre o Software Development Kit (SDK) e Runtime Environment (JRE) o qual é um subconjunto do SDK. O Java Runtime Environment ou JRE é o software necessário para rodar qualquer aplicação desenvolvida na plataforma Java. Usuários finais comumente usam JRE em pacotes de softwares e plugins de navegadores. A Sun também distribui um conjunto do JRE chamado Java 2 SDK, mais comumente conhecido como JDK, o qual inclui ferramentas de desenvolvimento como o compilador do Java, Javdoc, e debugador (GUIDE, 2016).

Existiam 5 objetivos principais na criação do Java: deveria usar a metodologia de linguagem orientada a objeto; deveria permitir o mesmo programa ser executado em diferentes plataformas; deveria conter um sistema interno de suporte para conexão de computadores; deveria ser designado a executar o código remotamente com segurança; deveria ser de fácil uso considerando as partes boas de outras linguagens orientadas a objeto. Para chegar nesses objetivos de suporte à conexões e execução de códigos remota, programadores de Java as vezes acham necessário usar extensões como CORBA, Internet Communications Engine ou OSGi (GUIDE, 2016).

2.9 Aplicativos

Um aplicativo é qualquer programa, ou grupo de programas, que é desenvolvido para o usuário final. O objetivo dos aplicativos é empregar capacidades de um aparelho

eletrônico desempenhar uma tarefa a qual o usuário deseja. Aplicativos incluem programas de dados, editores de textos, navegadores de internet, entre outros (BEAL, 2016). Todos esses aplicativos podem ser identificados através de ícones na tela de seu dispositivo, para melhor ilustrar é possível observar a presença desses aplicativos na Figura 2.9.

Figura 2.9: Aplicativos

Fonte: Android Developers (2017).

Vários aplicativos, com funções relacionadas, juntos em um pacote as vezes são conhecidos como *application suite*. Há também os aplicativos para empresas que são desenvolvidos de acordo com as necessidades das organizações e para grande distribuição de dados em um ambiente grande. Outros tipos de aplicativos são: educacionais, para informação, desenvolvimento de mídia, entre outros (TECHOPEDIA, 2017).

Os aplicativos vêm se popularizando desde 2007 graças aos *smartphones*. Eles surgiram para trazer mais praticidades a tarefas feitas no dia a dia, e muitos deles tem papeis fundamentais na vida de muitas pessoas. Um dos primeiros aplicativos criados foi o jogo *snake*, introduzido em 1998 como um jogo pré baixado os aparelhos Nokia e isso se tornou uma sensação global.

2.10 Bluetooth

A origem do Bluetooth é de 1996. Em uma reunião, os representantes de Ericsson, Nokia e Intel discutiam sobre uma nova tecnologia. Quando a conversa se tornou sobre qual seria o nome desta nova tecnologia, Jim Kardash da Intel sugeriu “Bluetooth”, se referindo ao rei Dinamarquês do século X Harald Bluetooth Gormson que unificou a Dinamarca com a Noruega. O monarca tinha um dente azul morto (FENDELMAN, 2017).

O Bluetooth é uma tecnologia sem fio usada para transferir dados entre diferente aparelhos eletrônicos. Em vez de criar uma conexão local (LAN) ou em uma área ampla (WAN), o Bluetooth cria uma conexão pessoal (PAN) (FENDELMAN, 2017). A Figura 2.10 ilustra de maneira clara uma rede de informações *Bluetooth* e sua distância limitada

Figura 2.10: Rede Bluetooth

Fonte: Mundo Max (2017).

As caraterísticas chave para a tecnologia Bluetooth são: maior facilidade de operação, menor consumo de energia, disponível em valores mais baratos, robustez. O Bluetooth permite que você mantenha as mãos livres enquanto faz chamadas, permite fazer impressões sem cabos, e sincronização automática de PDA. Vários tipos de aparelhos utilizam a tecnologia Bluetooth estão disponíveis no mercado. Esses diferentes tipos são: computadores, rádios, fones de ouvido, mouses, teclados (MARY, 2017).

O Bluetooth também é utilizado na automação de casa, permitindo que as luzes, temperatura, janela, portas, sistemas de segurança, entre outros sejam controlados por meio de computadores, celulares. A maioria dos fabricantes de automóveis oferece a tecnologia Bluetooth em seus produtos. O Bluetooth permite, com o reconhecimento de voz, que sejam enviadas e recebidas mensagens. Além disso, com o Bluetooth, é possível controlar o rádio o veículo, tocar a música do celular no rádio do veículo. Na saúde, médicos utilizam o Bluetooth em medidores de glicose, medidores de batimento cardíaco, medidores de pressão, inaladores para asma entre outros produtos que monitoram as condições do paciente (FENDELMAN, 2017).

O Bluetooth v2.1 possui alcance de até 100 metros (ambiente aberto), frequência entre 2.402 e 2.481 GHz, taxa máxima de transmissão entre 1 e 3 Mbit/s e padrão IEEE 802.15.1 (SPONAS, 2016).

2.11 Android Studio

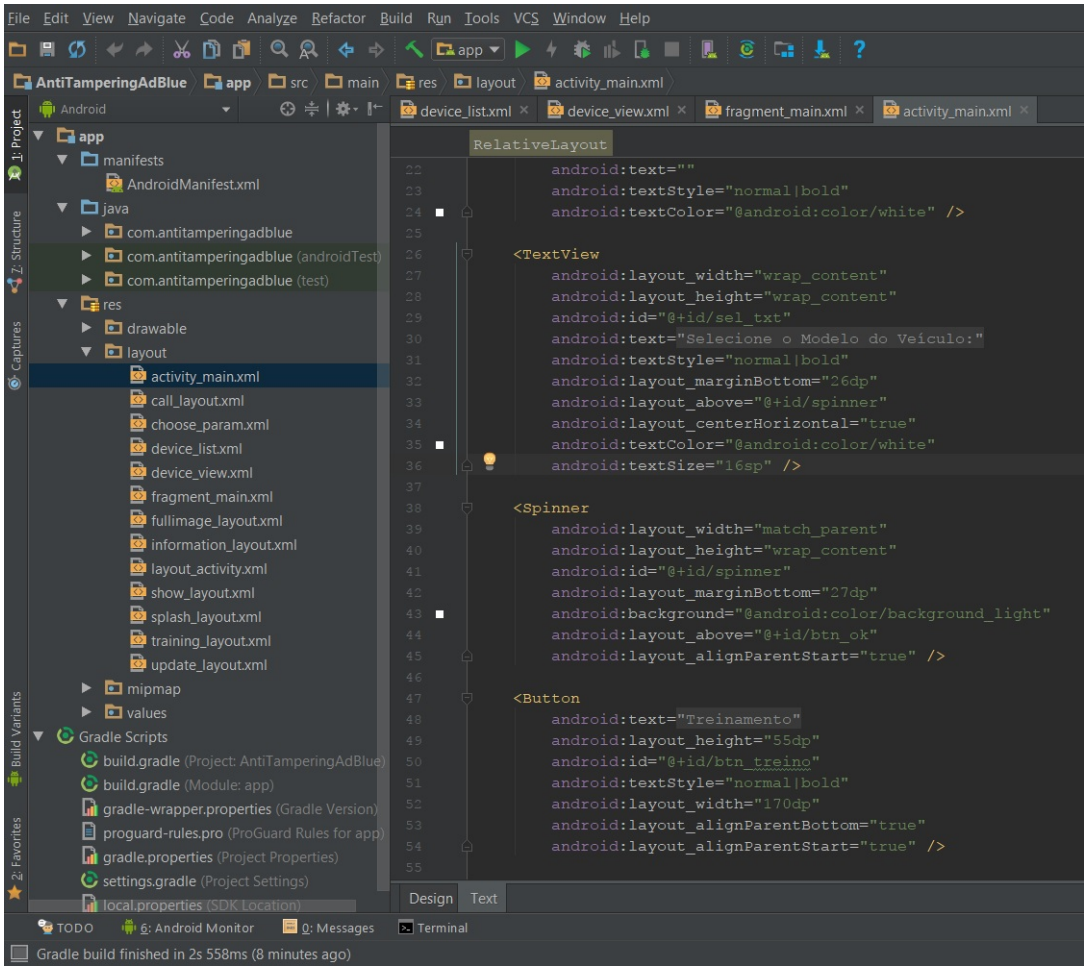
Há vários modos de desenvolver para Android, porém a oficial e mais poderosa é com o Android Studio. Desenvolvido pela Google e usada pela maioria dos aplicativos que são usados no dia a dia.

O Android Studio foi primeiramente anunciado na conferência da Google I/O em 2013 e foi lançado para o público em 2014 após várias versões beta. Antes do seu lançamento, o desenvolvimento para Android era predominantemente feito pela plataforma de desenvolvimento do Eclipse, a qual é mais genérica que a plataforma do Java que também suporta inúmera outras linguagens de programação (MULLIS, 2017).

O objetivo do Android Studio é fornecer uma interface para criação de aplicativos para que se possa lidar com complicadas administrações de dados por trás da interface (MULLIS, 2017). Essa interface pode ser observada na Figura 2.11, e é possível notar que o sistema de pastas na parte esquerda da figura facilita muito na hora de desenvolver um aplitativo, pois estas pastas são divididas de maneira intuitiva entre pastas envolvendo *layout*, pastas envolvendo as classes java e pastas envolvendo

outras aplicações como inserção de imagens e definições de acesso e de telas na parte de cima na pasta chamada *manifest*.

Figura 2.11: Interface Android Studio



Fonte: O Autor (2018).

O Android Studio torna a vida relativamente mais fácil se comparado com outros softwares não especializados, mas há também um caminho a ser percorrido para poder usufruir completamente de sua intuitividade e experiência mais polida. A linguagem de programação utilizada é o Java. O Android Studio dá acesso ao Android SDK ou Kit de Desenvolvimento de Software. Então, é como uma extensão ao Java que permite rodar suavemente em dispositivos Android e tem vantagem em hadware nativo. O Java é necessário para escrever programas, o Android SDK é necessário para fazer esses programas rodarem no Android e o Android Studio faz o trabalho de uni-los. Ao mesmo tempo, o Android Studio torna disponível rodar o código tanto em um emulador como

em um hardware ligado a máquina. Também é possível debugar o programa enquanto ele roda, tendo acesso ao *feedback* de erros para, assim, poder resolver problemas mais rapidamente (MULLIS, 2017).

Outra parte interessante desse *software* é que ele é muito bem organizado em termos de desenvolvimento e clareza de onde se encontram as principais abas em que se precisa trabalhar para o desenvolvimento de um aplicativo. O fato da codificação dos *layouts* ficarem bem próximos da codificação da lógica por trás dos elementos presentes em cada um desses *layouts* e da maneira com que essa codificação gráfica e lógica se complementam e se relacionam, torna a experiência de desenvolver para o Android Studio algo único.

2.12 Dongles

Hoje em dia, os *dongles* estão se tornando cada vez mais comuns em nossas vidas. Seja para conectar seu *laptop* à TV, monitorar dados seus ou de outras pessoas, ou até monitorar dados de veículos, todas essas funcionalidades necessitam de um *dongle*.

Um *dongle* é um pequeno *hardware* que pode se conectar a outro dispositivo para prover funcionalidades adicionais a ele. Na computação, o termo é primariamente associado com *hardwares* que fornecem um mecanismo de cópia protegida para *softwares* proprietários, nos quais o *dongle* deve ser conectado ao sistema em que o *software* está instalado para poder prover uma função (WIKIPEDIA, 2015).

Os primeiros *dongles* eram conectados aos computadores da IBM (*International Business Machines*) via DB-25 para prevenir uso sem autorização de *softwares* proprietários enquanto ainda permitia o uso normal da impressora. O termo "*dongle*" também está associado aos dispositivos similares que provêm meios adicionais de conexão *wireless* a dispositivos (com suporte para *Wi-Fi* ou *Bluetooth*) (WIKIPEDIA, 2015).

2.12.1 Dongle Simma Software

Para a aplicação envolvida neste projeto foi decidido o uso de um *dongle* para facilitar e tornar mais intuitivo essa interface entre veículo e celular e também poder realizar a aquisição dos dados no barramento e transmissão deles através da comunicação *Bluetooth*. O dispositivo encontrado para realizar essa intermediação foi o *dongle* VNA-232 da empresa *Simma Software* que pode ser visto na Figura 2.12.

Figura 2.12: Dongle Simma Software

Fonte: Simma Software (2016).

Este é um adaptador de rede veicular de alto desempenho e baixa latência com suporte para conexão *Bluetooth*. O dispositivo da Figura 2.12 suporta as interfaces CAN e J1939, com detecção automática de taxa de transmissão, e uma única interface J578/J1708. Todas as interfaces são certificadas para perda de pacotes nula e para operar com carga total no barramento lendo as mensagens presentes no veículo e podendo transmití-las através do *Bluetooth* para posterior análise (SOFTWARE, 2016).

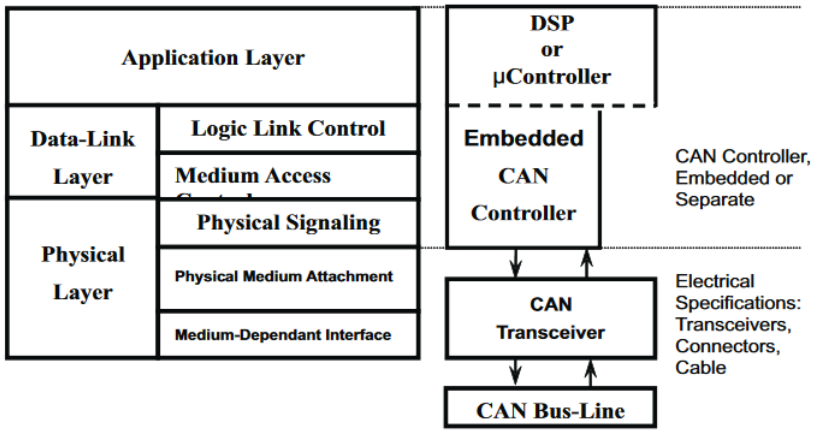
2.13 Protocolo CAN

O barramento CAN foi desenvolvido pela BOSCH como um sistema de transmissão de mensagens de múltiplos mestres, utilizando uma taxa de transmissão máxima de 1 Mbps. Ao contrário de uma rede tradicional como o USB ou Ethernet, a rede CAN não envia blocos de dados enormes do ponto A para um ponto B sob a supervisão de um

mestre que controla o barramento. Em uma rede CAN, muitas mensagens curtas, como a temperatura ou RPM são transmitidas para toda a rede, o que provê consistência de dados em cada um dos nós do barramento (INSTRUMENTS, 2016).

CAN é uma comunicação serial de barramentos definida pela ISO (*International Standardization Organization*) desenvolvida pela indústria automotiva para substituir a complexa cablagem por um conjunto de dois cabos do barramento. A especificação necessita de uma alta imunidade a interferências elétricas e a habilidade de se auto diagnosticar e reparar os erros dos dados. Essas características fizeram com que a rede CAN obtivesse popularidade em diversos setores da indústria como por exemplo o de automação, médico e manufatura (INSTRUMENTS, 2016). A arquitetura ISO 11898 define as duas camadas OSI mais baixas como podem ser vistas na Figura 2.13.

Figura 2.13: Arquitetura Padrão ISO



Fonte: Texas Instruments (2016).

O protocolo de comunicação CAN, ISO-11898: 2003, descreve como a informação é passada entre dispositivos em uma rede em conformidade com o modelo OSI (*Open System Interconnection*) definido em diversas camadas. A comunicação real entre dispositivos conectados por um meio físico é definida pelo modelo da camada física (INSTRUMENTS, 2016).

A camada de aplicação estabelece a comunicação para um nível superior de aplicação de protocolo específico como o independente *CANopen*. Esse protocolo é su-

portado pelos usuários internacionais e grupos de manufatura, CiA (*CAN in Automation*) (INSTRUMENTS, 2016).

2.13.1 CAN Normal e CAN Estendida

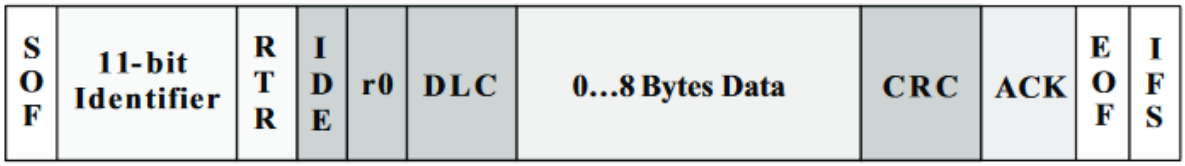
O protocolo de comunicação CAN é um protocolo de múltiplo acesso com detecção de colisão e arbitrariedade em detecção de prioridade de mensagem (CSMA/CD+AMP). CSMA significa que cada nó no barramento precisa aguardar por um certo período de inatividade antes de tentar enviar uma mensagem. CD+AMP significa que as colisões são tratadas através de uma organização arbitrária de bits, baseada em uma prioridade pré-programada de cada mensagem em seu campo de identificação. O identificador de maior prioridade sempre ganha o acesso ao barramento. Isso é, o último identificador em alta continua transmitindo porque tem a maior prioridade. Já que cada nó em um barramento importa na hora de transcrever cada bit, o nó arbitrário sabe discernir se o nível lógico está em alta ou não, tratando cada caso de acordo com o procedimento necessário (INSTRUMENTS, 2016).

O padrão ISO 11898:2003, com o identificador padrão de 11 bits, provê taxas de transmissão de sinais de 125kbps a 1 Mbps. Posteriormente, foi criado um padrão com um identificador de 29 bits. O identificador de 11 bits que pode ser visto na Figura 2.14, provê espaço para 2048 identificadores de mensagens. Enquanto isso, o padrão estendido de 29 bits, que pode ser visto na Figura 2.15 provê espaço para 537 milhões de identificadores (INSTRUMENTS, 2016).

Além disso, são compostos por muitos bits com significados específicos e também um campo específico apenas para realizar o *checksum* para identificação e posterior tratamento de eventuais erros na rede (INSTRUMENTS, 2016).

2.13.2 CAN Normal

Figura 2.14: Estrutura CAN de 11 bits



Fonte: Texas Instruments (2016).

O significado dos bits da Figura 2.14 são:

SOF – O bit que representa o *Single Dominant Start Of Frame* marca o início de uma mensagem, e também é usado para sincronizar os nós em um barramento depois do tempo ocioso.

identificador - O identificador padrão de 11 bits estabelece a prioridade da mensagem. Quanto menor o valor binário, maior a prioridade.

RTR – O bit que representa o *Remote Transmission Request* é dominante quando se requer informação de outro nó. Todos os nós recebem uma requisição, mas o identificador que determina o nó específico. Os dados de resposta também são recebidos por todos os nós e usados apenas pelo que interessa.

IDE – O bit que representa o *Identifier Extension* é responsável por dizer se a mensagem está sendo transmitida com um identificador padrão ou estendido.

r0 – É o bit reservado para possível uso futuro.

DLC – Os 4 bits que representam o *Data Length Code* são responsáveis por identificar o número de bytes de dados sendo transmitidos.

CRC – Os 16 bits que representam o *Cyclic Redundancy Check* são responsáveis por realizar o *checksum* dos dados para detecção de erros.

ACK – Cada nó recebendo uma mensagem sobrescreve esse bit recessivo na mensagem original por um dominante, indicando que uma mensagem livre de erros foi enviada. Se o nó receptor identificar um erro e deixar esse bit recessivo, ele descarta a mensagem e o nó de envio repete a mensagem. Essa mensagem é repetida até que seja enviada sem nenhum problema e posteriormente seja possível a continuação da

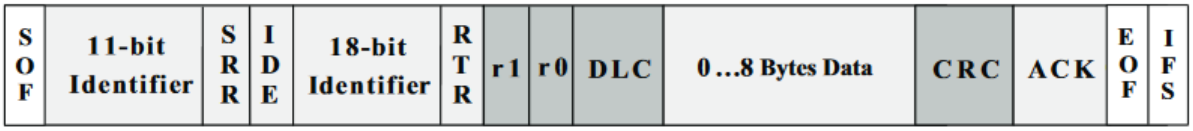
execução.

EOF – Os 7 bits que representam o *End Of Frame* são responsáveis por marcar o fim de um pacote CAN e desabilitar o emparelhamento de bits. Quando 5 bits ficam no mesmo nível lógico em sucessão durante operação normal, um bit de nível lógico oposto é emparelhado ao lado.

IFS – Os 7 bits que representam o *Interframe Space* são responsáveis por conter o tempo necessário para o controlador mover um pacote recebido de maneira correta para sua devida posição na área de *buffer*.

2.13.3 CAN Estendida

Figura 2.15: Estrutura CAN de 29 bits



Fonte: Texas Instruments (2016).

Como visto na Figura 2.15, a CAN estendida e muito parecida com a CAN normal, porém as principais diferenças para o modelo estendido foram a adição dos seguintes elementos:

SRR - O bit que representa o *Substitute Remote Request* é responsável por substituir o RTR da mensagem normal.

r1 – Seguindo o RTR e os bits do r0, um bit reserva adicional foi incluído depois do bit de DLC.

2.13.4 Tipos de Mensagens

Os quatro tipos de mensagens, ou pacotes, que podem ser transmitidos em um barramento CAN são os pacotes de dados, pacotes remotos, pacotes de erros e pacotes de sobrecarga.

2.13.4.1 Pacote de Dados

O pacote de dados é o tipo mais comum de mensagem, e ele compara o campo arbitrário, o campo de dados, o campo CRC e o campo de reconhecimento. O campo arbitrário possui um identificador de 11 bits como na Figura 2.14 e o bit RTR, que é dominante para o pacote de dados. Na Figura 2.15, contém o identificador de 29 bits e o bit de RTR. O próximo é o campo de dados que contém 0 a 8 bytes de dados, e o campo CRC que contém os 16 bits de *checksum* usados para detecção de erros. Por último vem o campo de reconhecimento (INSTRUMENTS, 2016).

2.13.4.2 Pacote Remoto

O propósito do pacote remoto é o de solicitar a transmissão de dados de outro nó. O pacote remoto é parecido com o de dados, mas possui duas diferenças importantes. Primeiro, o tipo da mensagem é marcado explicitamente como pacote remoto por um RTR recessivo em um campo arbitrário, e segundo, não se transmite dados (INSTRUMENTS, 2016).

2.13.4.3 Pacote de Erros

O pacote de erros é uma mensagem especial que viola as regras de formatação das mensagens CAN. O transmissor de origem então retransmite automaticamente a mensagem. Um sistema que consiste no *checksum* da mensagem garante que o nó não seja capaz de ficar retransmitindo pacotes errados (INSTRUMENTS, 2016).

2.13.4.4 Pacote de Sobrecarga

O pacote de sobrecarga é executado para completar o ciclo. Ele é similar ao de erros em relação ao formato, e é transmitido por um nó que fica ocupado demais, ou seja, repleto se mensagens sem conseguir transmiti-las na mesma velocidade em que chegam. E é primariamente usado para realizar um atraso extra entre mensagens

(INSTRUMENTS, 2016).

2.13.4.5 Pacote Válido

Uma mensagem é considerada como sendo livre de erros quando o último bit do campo EOF de uma mensagem recebe um estado de livre de erros. Um bit dominante no EOF faz com que o transmissor repita a transmissão (INSTRUMENTS, 2016).

2.13.5 Checagem de Erros e Confinamento de Falhas

A robustez da rede CAN pode ser atribuída em parte aos seus procedimentos abundantes de verificação de erros. O protocolo CAN incorpora cinco métodos de verificação de erros: três no nível da mensagem e dois no nível do bit. Se uma mensagem falhar em qualquer um desses métodos de detecção de erros, ela não será aceita e um pacote de erros será gerado a partir do nó receptor. Isso força o nó de transmissão a reenviar a mensagem até que seja recebida corretamente. No entanto, se um nó defeituoso desligar o barramento repetindo continuamente a transmissão de um erro, sua capacidade de transmissão será removida pelo controlador após o limite de erro ser atingido (INSTRUMENTS, 2016).

A verificação de erros no nível da mensagem é reforçada pelos campos CRC e ACK exibidos na Figura 2 e na Figura. O CRC de 16 bits contém o *checksum* da aplicação anterior de dados para detecção de erros com uma soma de verificação de 15 bits e um delimitador de 1 bit. O campo ACK tem dois bits de tamanho que consiste no bit de reconhecimento e um bit delimitador de reconhecimento. Essa verificação procura por campos na mensagem que devem sempre ser bits recessivos. Se um bit dominante for detectado, um erro será gerado. Os bits verificados são SOF, EOF, delimitador ACK, e os bits delimitadores CRC (INSTRUMENTS, 2016).

A nível de bit, cada bit transmitido é monitorado pelo transmissor da mensagem. Se um bit de dados (não bit arbitrário) for gravado no barramento e seu oposto for lido, um erro será gerado. As únicas exceções para isso são com o campo de identificador de

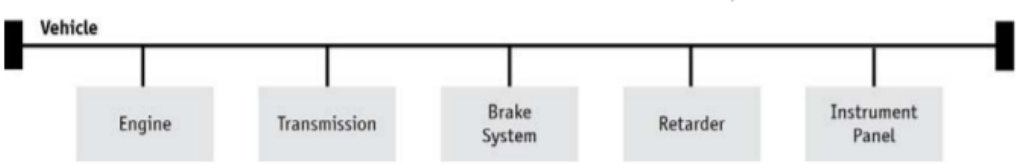
mensagem que é usado para arbitráriedade e o slot de reconhecimento que requer que um bit recessivo seja substituído por um bit dominante. O método final de detecção de erros é a regra de preenchimento de bits, em que após cinco bits consecutivos do mesmo nível lógico, se o próximo bit não for um complemento, será gerado um erro. O emparelhamento garante que as bordas de súbida estejam disponíveis para sincronização contínua da rede. Ele também garante que um fluxo de bits não seja confundido com um pacote de erros ou o espaço entre pacotes de sete bits que significa o final de uma mensagem. Bits emparelhados são removidos pelo controlador de um nó receptor antes que os dados sejam encaminhados para a aplicação (INSTRUMENTS, 2016).

Com essa lógica, um pacote de erro ativo consiste em 6 bits dominantes, violando a regra de emparelhamento de bits. Isso é interpretado como erro por todos os nós da CAN que então geram seus próprios pacotes de erro. Isso significa que um pacote de erros pode ser originado dos 6 a 12 bits dentre todas as respostas. Esse pacote de erros então é mandado seguido de um campo delimitador de 8 bits recessivos e um período ocioso de barramento antes da mensagem corrompida ser retransmitida. É importante notar que a mensagem retransmitida ainda precisa atender ao critério arbitrário do barramento (INSTRUMENTS, 2016).

2.14 Protocolo J1939

O SAE J1939 é usado na área de veículos comerciais para comunicação deles. O SAE J1939 usa o CAN (Controller Area Network, ISO11998) como camada física. É uma prática recomendada que define quais e como os dados são comunicados entre as Unidades de Controle Eletrônico (ECU) dentro da rede de um veículo (VECTOR, 2010). Controladores típicos são o Motor, Freio, Transmissão, assim como podem ser vistos na Figura 2.16.

Figura 2.16: Típica Rede J1939 em Veículos



Fonte: VECTOR (2010).

As características particulares do J1939 são: identificador estensivo da rede CAN (29 bits), taxa de transmissão de 250 kbit/s, comunicação *Peer-to-Peer* e *broadcast*, protocolos de transporte para até 1785 bytes de dados, gerênciamento de rede, definição de grupos de parâmetros para veículos comerciais, ferramentas de diagnose.

Existem vários padrões derivados do SAE J1939. Esses padrões usam os recursos básicos do SAE J1939 com um conjunto diferente de grupos de parâmetros e camadas físicas modificadas. Esses padrões serão vistos na sequência:

2.14.1 ISO11783

Define a comunicação entre o trator e os implementos em um barramento do implemento. Ele especifica alguns serviços na camada de aplicação, como o Terminal Virtual, a ECU e o Servidor de Arquivos. Adiciona um protocolo de transporte estendido e gerenciamento de conjunto de trabalho (VECTOR, 2010).

2.14.2 NMEA2000

Define grupos de parâmetros para a comunicação entre dispositivos marítimos. Especifica o protocolo de transporte adicional de pacotes rápidos (VECTOR, 2010).

2.14.3 ISO11992

Especifica a troca de informações entre o veículo de via e o veículo rebocado. Ele usa o mesmo formato de grupo de parâmetros que o J1939 em uma camada física diferente com 125 kbit/s (VECTOR, 2010).

2.14.4 FMS

O padrão FMS (*Fleet Management System* define um *gateway* entre a rede de veículos J1939 e um sistema de gerenciamento de frota (VECTOR, 2010).

2.14.5 Grupo de Parâmetros

Um grupo de parâmetros é um conjunto de parâmetros pertencentes ao mesmo tópico e que compartilham a mesma taxa de transmissão. O comprimento de um grupo de parâmetros não está limitado ao comprimento de um pacote CAN. Normalmente, um grupo de parâmetros tem um comprimento mínimo de 8 bytes até 1785 bytes. Grupos de parâmetros com mais de 8 bytes requerem um protocolo de transporte para transmissão (VECTOR, 2010).

2.14.6 Interpretação do Identificador CAN

O identificador CAN de uma mensagem J1939 contém o Número de Grupo de Parâmetros (PGN), endereço de origem, prioridade, bit de página de dados, bit de página de dados estendida e um endereço de destino (somente para um PG *peer-to-peer*) (VECTOR, 2010).

O identificador é composto de acordo com a Figura 2.17 da seguinte forma:

Figura 2.17: Estrutura do Identificador

Priority	Extended Data Page	Data Page	PDU Format	PDU Specific	Source Address
3 bit	1 bit	1 bit	8 bit	8 bit	8 bit

Fonte: VECTOR (2010).

2.14.7 Número do Grupo de Parâmetros

Cada grupo de parâmetros é endereçado por um número único - o PGN. Para o PGN, é utilizado um valor de 24 bits composto pelos 6 bits definidos como 0, formato PDU (8

bits), PDU específico (8 bits), página de dados (1 bit) e página de dados estendida (1 bit) (VECTOR, 2010).

Existem dois tipos de números de grupos de parâmetros. O primeiro são os PGNs globais, eles identificam grupos de parâmetros que são enviados para todos (*broadcast*). Aqui, o formato PDU, PDU específico, página de dados e página de dados estendida são usados para identificação do grupo de parâmetros correspondente. Em PGNs globais, o formato PDU é 240 ou maior e o campo específico de PDU é uma extensão de grupo (VECTOR, 2010).

O segundo são os PGNs específicos são para grupos de parâmetros enviados a dispositivos específicos (ponto a ponto). Aqui, o formato PDU, página de dados e página de dados estendido são usados para identificação do grupo de parâmetros correspondente. O formato da PDU é 239 ou menos e o campo específico da PDU está definido como 0 (VECTOR, 2010).

2.14.8 Número de Parâmetro Suspeito

Um número de parâmetro suspeito é atribuído a cada parâmetro de um grupo de parâmetros ou componente. Ele é usado para fins de diagnóstico para relatar e identificar operações anormais de uma aplicação de controlador (CA). O SPN é um número de 19 bits e tem um intervalo de 0 a 524287. Para parâmetros proprietários, um intervalo de 520192 a 524287 é reservado (VECTOR, 2010).

2.14.9 Grupo de Parâmetros Especiais

2.14.9.1 Grupo de Requisição de Parâmetros

O grupo de requisição de parâmetros (RQST, PGN 00EA0016) pode ser enviado a todos ou a uma CA específico para solicitar um grupo de parâmetros especifico. O RQST contém o PGN do grupo de parâmetros de solicitação. Se o receptor de uma solicitação específica não puder responder, deverá enviar uma confirmação negativa. O RQST tem um código de comprimento de dados de 3 bytes e é o único grupo de

parâmetros com um código de comprimento de dados menor que 8 bytes (VECTOR, 2010).

2.14.9.2 Grupo de Reconhecimento de Prâmetros

O grupo de parâmetros de reconhecimento (ACKM, PGN 00E80016) pode ser utilizado para enviar uma confirmação negativa ou positiva, isto é, em resposta a um pedido (VECTOR, 2010).

2.14.9.3 Grupo de Parâmetros de Solicitação de Endereço

O grupo de parâmetros de solicitação de endereço (ACL, PGN 00EE0016) é usado para gerenciamento de rede (VECTOR, 2010).

2.14.9.4 Grupo de Parâmetros de Protocolo de Transporte

Os grupos de parâmetros de protocolo de transporte (TPCM, PGN 00EC0016 e TPDT, PGN 00EB0016) são usados para transferir grupos de parâmetros com mais de 8 bytes de dados (VECTOR, 2010).

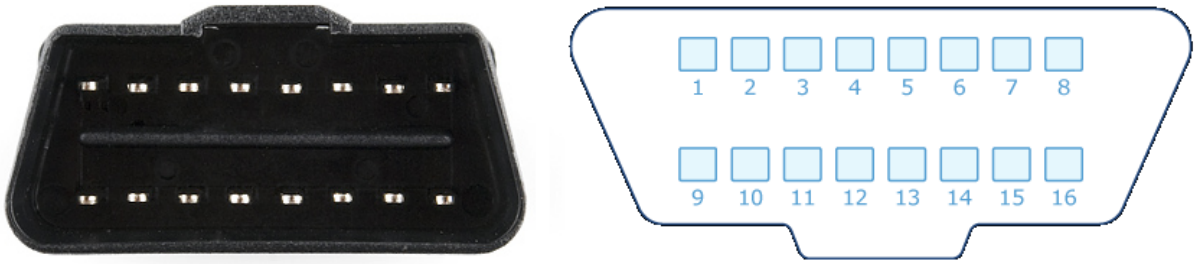
2.15 Protocolo OBD-II

OBD significa ”On-Board Diagnostics”e se refere à um sistema originalmente criado para reduzir emissões monitorando o desempenho de diversos componentes do motor. Em 1980 a primeira geração de sistemas OBD foi criada, mas na época eram utilizados conectores, *hardwares* e protocolos proprietários, ou seja, se o usuário desejasse utilizar o sistema, devería comprar diversos equipamentos para cada veículo. Já em 1990 a SAE (*Society of Automotive Engineers*) e a ISO (*International Standardization Organization*) decretaram diversas regras que descreviam a troca de informações digitais entre ECUs e também o uso de um conector de diagnose padrão (SAEJ1962) e

que se comunicassem via um dos protocolos padrões do sistema OBD-II (SOLUTIONS, 2016).

O OBD-II provê acesso às informações referentes aos DTCs (*Diagnostic Trouble Codes*) que tenham relação com o *Powertrain* (Motor e transmissão) e ao sistema de controle de emissões. Também é possível acessar outras informações tais como: VIN (*Vehicle Identification Number*), número de identificação de calibração, contador de ignição e do sistema de controle de emissões. Existem diversos códigos de falhas que tratam de muitas partes do veiculo, seja do sistema de *powertrain*, chassis ou barramento (BARRETO, 2017). Para o acesso de todas essas informações é necessário utilizar o conector SAEJ1962, conhecido como conector OBD-II e mostrado na Figura 2.18.

Figura 2.18: Conector OBD-II



Fonte: Sparkfun (2015).

Todos os pinos do conector possuem funções previamente regulamentadas pelas regras estabelecidas pela ISO e SAE, o mapeamento do conector é o seguinte:

- **Pino 1:** Opção do Fabricante;
- **Pino 2:** SAE J850 +;
- **Pino 3:** Opção do Fabricante;
- **Pino 4:** GND;
- **Pino 5:** GND;
- **Pino 6:** CAN high (ISO 15765-4 e SAE J2284);

- **Pino 7:** ISO 9141-2 K Line;
- **Pino 8:** Opção do Fabricante;
- **Pino 9:** Opção do Fabricante;
- **Pino 10:** SAE J850 -;
- **Pino 11:** Opção do Fabricante;
- **Pino 12:** Opção do Fabricante;
- **Pino 13:** Opção do Fabricante;
- **Pino 14:** CAN low (ISO 15765-4 e SAE J2284);
- **Pino 15:** ISO 9141-2 L Line;
- **Pino 16:** Tensão da Bateria.

O OBD-II possui 10 modos padronizados para adquirir as informações de diagnose para emissões. Porém, esses 10 modos não são o suficiente. Com base nessa falta foi criado o chamado UDS (*Unified Diagnostic Services*), para enriquecer os dados disponíveis. Cada montadora possui seus próprios PIDs (*Parameter IDs*) implementados via modos UDS (BARRETO, 2017).

Portanto, para adquirir os parâmetros da ECU, são utilizados esses parâmetros chamados de PIDs. Eles são classificados por modos de operação, definidos pela SAE J1979 / ISO 15765-4. Ao todo são 10 PIDs (SAE, 2017).

- **Service $01:** *Request Current Powertrain Diagnostic Data*;
- **Service $02:** *Request Powertrain Freeze Frame Data*;
- **Service $03:** *Request Emission-Related Diagnostic Trouble Codes*;
- **Service $04:** *Clear/Reset Emission-Related Diagnostic Information*;
- **Service $05:** *Request Oxygen Sensor Monitoring Test Results*;

- **Service $06:** *Request On-Board Monitoring Test Results for Specific Monitored Systems*;
- **Service $07:** *Request Emission-Related Diagnostic Trouble Codes Detected During Current or Last Completed Driving Cycle*;
- **Service $08:** *Request Control of On-Board System, Test or Component*;
- **Service $09:** *Request Vehicle Information*;
- **Service $0A:** *Request Emission-Related Diagnostic Trouble Codes with Permanent*.

Assim, ao fazer uso dos PIDs, é possível detectar falhas decorrentes dos sistemas que são relevantes para o trabalho, no caso os PIDs $03, $07 e $09, onde, para fins de gerar uma diagnose mais confiável com abrangência, esses dados serão utilizados em conjunto com aqueles analizados no próprio barramento CAN.

2.16 Canalyzer

No ano de 1992, a empresa Vector lançou a ferramenta CANalyzer. Foi o primeiro produto da companhia. A ferramenta de *software* Baseada em PC analisa a rede de comunicação CAN e pode prover várias janelas de análise e também arquivos de gravações de medições (NEWSLETTER, 2012). A introdução da camada do protocolo de dados CAN instituída em 1987 criou diversas oportunidades para o uso de eletrônicos no setor automotivo. Dentro de apenas alguns anos, esses eletrônicos passaram a suportar sistemas de auxilio ao condutor e diversas características. Os requisitos técnicos para analisar as comunicações por barramento aumentaram imensamente. Dr. Helmut Schelling, Martin Litschel e Eberhard Hinderer, convencidos do enorme potencial da rede CAN, em 1988 desenvolveram o primeiro *software* do mundo de análise do barramento CAN. O CANalyzer foi lançado para uso e continua sendo muito usado até os dias de hoje (NEWSLETTER, 2012).

Atualmente, o CANalyzer suporta alguns protocolos de alto nível, como CANopen, J1939 e KWP2000. O *software* também possui suporte a outras tecnologias de comunicação, tais como, Ethernet, Flexray, LIN e Most. Ele é um *software* compreensível com operações intuitivas para análise e simulação de comunicações de barramentos. É possível usá-lo para verificar qual o tipo de comunicação está ocorrendo no barramento. Também pode ser usado para enviar ou gravar dados. Para cada aplicação existem diversas funções básicas para iniciantes, bem como complexas e detalhadas para usuários experientes (VECTOR, 2010).

As três operações mais comuns realizadas no CANalyzer são: Analise e monitoramento do tráfego na rede, implementação de protocolos e análise de fluxo de dados, desenvolvimento de ECUs. Quanto à primeira operação, é possível monitorar o tráfego da rede de comunicação e fazer uso das possibilidades de análise do CANalyzer, como a janela de rastreio ou janelas de gráficos ou gravações de dados. É possível também desenvolver uma ECU e testar o seu *software* ao colocar a ECU em um estado definido. É possível testar esse sistema em seus limites. Para fazer isso, é preciso aumentar a carga do barramento de maneira incremental (VECTOR, 2013).

2.17 Trennadapter

No ramo automotivo, principalmente em empresas que trabalham com testes e *software* de ECU, é de suma importância que esses testes sejam feitos da maneira correta, seja para validar alguma atualização nesse *software*, ou inserção de um *software* novo em ECUs recém saídas de fábrica para serem alocadas em modelos novos de veiculos ou até simulações em laboratório com a utilização de ECUs avulsas possuídas pelas próprias empresas.

Uma característica muito importante a ser levada em conta é que cada modelo de veiculo, no caso deste projeto, mais especificamente caminhões, possui sua própria ECU. Por exemplo, todos os veiculos da marca 'X' produzidos entre o ano de 2010 a 2014 terão um certo modelo de ECU em seus motores, já os veiculos produzidos

na mesma época por uma marca ’Y’ terão outro modelo. Ou seja, existem muitos modelos de ECU presentes no mercado de caminhões atual sendo divididos por ano de fabricação e também marca, e cada um desses modelos possui conectores diferentes para acoplá-los ao motor e esses conectores são compostos por diversos pinos que são responsáveis por acessar múltiplas áreas do veiculo. Seja o barramento can, bateria, pedal do acelerador, injetores, etc. Um exemplo desses conectores está presente na Figura 2.19.

Figura 2.19: Conectores ECU

Fonte: Locanto (2018).

Como pode ser visto pela Figura 2.19, são muitos pinos e quando se está realizando testes específicos ou se deseja adquirir apenas os sinais da rede CAN, acaba por ser difícil o acesso a apenas alguns pinos. Um grande problema que surgiu quando se iniciaram os trabalhos em que era necessário acessar a ECU para realizar alterações de *software*, testes diversos em regime de operação e monitoramento era o fato de que para acessar os pinos específicos era preciso fazer uso de muitos *jumpers* alocados em cada um dos pinos e esse processo, quando o acesso de muitos pinos simultâneos se fazia necessário, passava a ser muito delicado e suscetível a influências externas como vibração, que prejudicava as conexões dos pinos com os *jumpers*.

Após algum tempo tendo esses procedimentos citados acima sendo feitos à base

de *jumpers*, e tendo em vista todos os problemas e fragilidade dos testes, surgiram os *trennadapter*. Equipamentos pesados compostos por diversos terminais tendo a mesma composição, conectores e mapeamento dos pinos da ECU equivalentes ao seu modelo. Ou seja, para cada modelo de ecu cujo conector difere, existe um *trennadapter* equivalente. Então ao realizar o acoplamento deste equipamento à sua ECU equivalente, é possível acessar os terminais daquela ECU de maneira simples através de um painel muito mais acessível. O que torna os testes e todos os procedimentos de campo e também laboratoriais realizados com as ECUs mais práticos e confiáveis, tendo em vista que o acoplamento do *trennadapter* à ECU é realizado por um conector fêmea que se acopla perfeitamente sem perigo de haver alguma desconexão por vibração ou outras influências externas. O equipamento *trennadapter*, ou 'caixa de terminais' pode ser visto na Figura 2.20.

Figura 2.20: Trennadapter

Fonte: Smart (2017).

CAPÍTULO 3

PROJETO

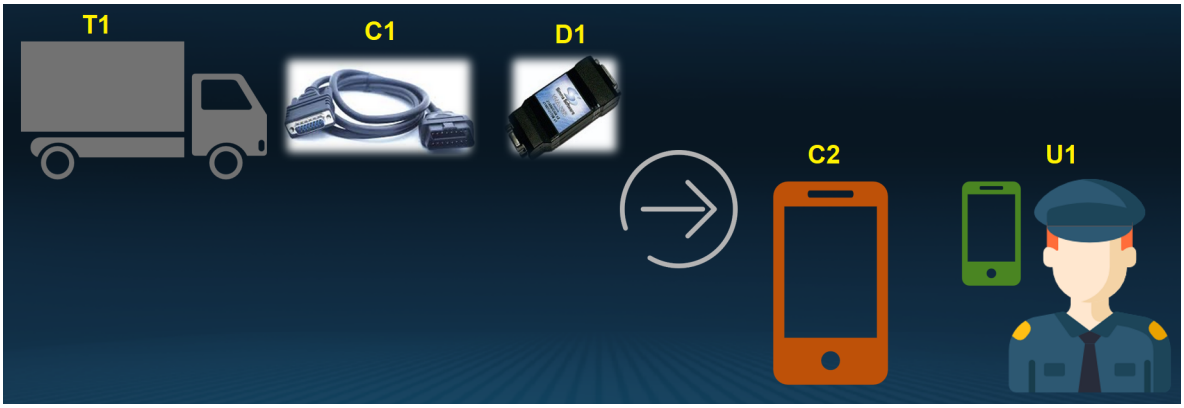
Este capítulo contemplará as informações relacionadas às metodologias, especificações e recursos necessários para o desenvolvimento deste trabalho. As definições seguirão feitas à partir da arquitetura geral, seguindo para funcionalidades envolvidas e por fim o porque e quais recursos de *software* e *hardware* foram optados.

3.1 Arquitetura Geral

Essa seção busca descrever, baseando-se no objetivo geral e específicos, a disposição de como o sistema do trabalho está projetado, tendo como base uma visão funcional. As explicações de *software* e *hardware* serão realizadas na seção 3.3.

A arquitetura modular do sistema, presente na Figura 3.1, foi elaborada tendo como base os principais pontos citados no objetivo geral em relação ao sistema capaz de adquirir, interpretar e transmitir dados do motor através do conector OBD, comunicação *Bluetooth*, protocolo SAE J1939 e barramento CAN.

Figura 3.1: Arquitetura Modular do Sistema



Fonte: O Autor (2018).

No canto superior esquerdo, é possível observar o caminhão (T1), essencial para a aplicação do projeto, pois ele que é o elemento do sistema em que são realizadas as fiscalizações e também é onde se situam todos os sistemas veiculares e onde teremos acesso aos sistemas relacionados à emissões, o motor, a ECU, o conector OBD-II e o barramento CAN, para análise de possível burla ou não do sistema de pós-tratamento.

Logo ao lado direito está o cabo (C1), responsável por realizar a interface de transferência dos dados obtidos no barramento do caminhão através do conector OBD-II até a outra extremidade, onde está localizado o conector referente ao dispositivo de aquisição e transmissão dos sinais.

O meio decidido para adquirir os dados provenientes do caminhão foi através de um *dongle* (D1), responsável por adquirir todas as informações que trafegam pelo barramento do veiculo, aglomerá-las e mandá-las por *Bluetooth*.

Devido à proposta do projeto ter como base um sistema além de tudo amigável para o usuário, no caso a PRF e os agentes do IBAMA, optou-se pela interface ser desenvolvida em formato de aplicativo, onde seria possível desenvolver algo intuitivo, que os usuários não estariam totalmente alheios e fosse fácil de aprender a utilizar para não requerer a presença de engenheiros no local. Tendo isso em vista, o *smartphone* (C2) é responsável por conter o *software* desenvolvido pelo autor para adquirir os dados enviados pelo *dongle* e informar ao usuário (U1) um diagnóstico confiável e que de um bom embasamento para futuras conclusões sobre os casos de burla do sistema de pós-tratamento.

Com as explicações dadas, é possível contemplar como o sistema deve operar, tendo em vista que os caminhões (T1) são todos os veiculos deste porte circulando nas rodovias do país que forem submetidos às fiscalizações. O cabo (C1) definido com uma extremidade contendo o conector do OBD-II e na outra compatível com o *dongle* (D1) escolhido, fornecido pela empresa *Simma Software*. Ainda não foi definido se o *smartphone* (C2) será fornecido junto com a solução ou se o policial poderá adquirir o aplicativo através de seu próprio celular.

3.2 Funcionalidades

Nesta etapa serão definidas as funcionalidades de cada um dos elementos descritos da arquitetura modular do sistema presente na seção 3.1, ou seja, qual o dever de cada um dos elementos do sistema em um nível mais intrínseco. Para auxiliar essa visão foi feito uma nova arquitetura, a arquitetura funcional do sistema que pode ser vista na Figura 3.2

Figura 3.2: Arquitetura Funcional do Sistema

Fonte: O Autor (2018).

No canto superior esquerdo, a primeira ilustração (M1) representa a um nível funcional, qual o propósito do caminhão no sistema, que é o de ao ser ligado, ter todos os seus sistemas eletroeletrônicos funcionando de maneira correta para que ao acessar seu barramento, seja possível extrair todas as informações que trafegam em sua rede CAN encapsuladas no protocólo SAE J1939 e também as informações referentes ao OBD-II.

Ao lado, a ilustração (T1) mostra uma transferência de dados, o que se refere exatamente à função do cabo, ilustrado na parte modular do sistema, que é a transferência de todos os dados obtidos a partir do barramento e do OBD-II do veiculo para o *dongle*.

Como todos esses dados estão sendo acumulados no dongle, a ilustração (T2) representa o conjunto de todos esses dados, que no caso do dispositivo optado e fornecido pela empresa *Simma Software* possui um tratamento das informações em que todos os vetores de dados, em formato SAE J1939 e OBD-II, são concatenados

em ordem de chegada para que seja formado um grande vetor contendo tudo e assim que o intervalo de transmissão for atingido, utilizar seu módulo *Bluetooth* para enviar todo esse vetor ao dispositivo com o qual estiver pareado.

A ilustração (B1) demonstra a funcionalidade de transmissão de dados optada pelo projeto, que no caso foi a transmissão através de *Bluetooth*, pelo fato do usuário estar próximo ao transmissor e também à facilidade de uso. E por fim, a ultima funcionalidade é referente à ilustração (D1), representando todo o software e suas etapas de aquisição do vetor aglomerado de dados, os procedimentos para transformá-los novamente em respectivas mensagens com estruturas iguais às anteriores, respeitando o estabelecido nos protocolos OBD-II e SAE J1939, e por fim a interpretação dos dados relevantes e exibição para o usuário através da interface do app.

3.3 Recursos

Nesta parte serão definidos quais os recursos utilizados na implementação do projeto em nível de *hardware* e *software*, bem como as justificativas do porque da escolha de cada uma das partes.

3.3.1 Recursos de Hardware

Toma-se como base a arquitetura modular exposta na seção 3.1, sendo possível observar a presença de três *hardware*: (C1), (D1) e (C2). Em relação aos cabos do sistema proposto, é necessário ter em mente qual o conector necessário para se adequar ao *dongle* escolhido e também estudar quais conectores são essenciais para a conexão com os caminhões a serem fiscalizados.

Sabe-se que após a evolução dos sistemas eletroeletrônicos nos veiculos, um ponto foi atingido onde seria necessário a implementação de algum meio de acesso à parte eletrônica do caminhão para testes e monitoramentos. Neste momento, nos anos 90, a ISO estabeleceu uma diretriz que relata a obrigatoriedade da inclusão de um conector seguindo os padrões do OBD-II, que pode ser visto na fundamentação teórica

do documento. Portanto, o cabo escolhido para o projeto foi aquele com extremidades compatíveis com o conector OBD-II e o *dongle*. Para melhor ilustrar, este cabo pode ser visto na Figura 3.3.

Figura 3.3: Cabo do Sistema

Fonte: VIP Pro Systems (2017).

Como o projeto está sendo desenvolvido em conjunto com a empresa e possui em seu termino como objetivo se tornar um KIT que possa ser vendido para os postos da PRF espalhados para o território nacional. Optou-se pela aquisição de um *dongle* já estabelecido no mercado e em funcionamento para que no momento das vendas, a parte do dispositivo seja apenas de aquisição e distribuição, focando principalmente no final na solução completa incluindo todos os elementos e principalmente o *software* desenvolvido. Ao definir que o *dongle* seria algum já presente no mercado, este teria que atender a alguns requisitos essenciais do projeto:

- Módulo *Bluetooth*;
- Tensão de 6 a 40 V;
- Suporte ao protocolo SAE J1939/CAN;
- *Baud Rate* de 250K a 500k;
- Programável;
- Garantia e fornecimento em larga escala;

- Detecção de colisão.

Após analise de todos esses requisitos e pesquisas online para buscar o modelo que mais se encaixasse nos parâmetros acima, o melhor modelo encontrado tanto em questão de atendimento dos requisitos quanto em questão de preço foi o *dongle* VNA-BT da empresa *Simma Software*. Vale salientar que além de atender todas as condições também foram realizadas reuniões com a equipe da empresa para melhor alinhamento de expectativas. O *dongle* em questão está ilustrado na Figura 3.4.

Figura 3.4: Dongle VNA-BT

Fonte: Simma Software (2017).

O último *hardware* envolvido no sistema é o *smartphone*, porém, ainda não foi decidido se ele será fornecido em conjunto com o KIT e já com o *software* integrado, ou se os usuários terão a opção de baixar o *software* com algum link exclusivo na hora da aquisição do KIT pelos seus próprios aparelhos celulares.

3.3.2 Recursos de Software

Os recursos de *software* necessários para o sistema proposto tem relação, principalmente, com o *smartphone* que será utilizado pelos usuários. É de suma importância que seja definido de maneira coesa, para que todos esses recursos, possam ser a fundamentação de todo o desenvolvimento da solução que será feita.

O principal passo ao desenvolver o projeto e após ter decidido que seria feito como uma solução de aplicativo, é o de optar pelo sistema operacional no qual a solução será aplicada. Os dois principais são o sistema Android e o sistema IOS, ambos possuem seus prós e contras e também suas linguagens especificas e ambientes de desenvolvimento distintos.

Na etapa de escolher com qual sistema operacional trabalhar, levou-se em consideração os seguintes itens:

- I. Custos de desenvolvimento para a plataforma;
- II. Custos de aquisição de dispositivos para teste e posterior distribuição [celular/-tablet];
- III. Custos e informações disponíveis de plataforma de desenvolvimento;
- IV. Abrangência de uso do sistema em território nacional;
- V. Linguagem de programação envolvida;
- VI. Familiaridade do desenvolvedor;
- VII. Familiaridade do sistema com soluções feita pela empresa.

Tendo em vista que o projeto precisa ser desenvolvido para a PRF e que se trata de um órgão público, e no fim precisará ser submetido à uma licitação, o preço do sistema tem que ser o mais barato possível. A partir dessas necessidades os itens I, II e III foram concebidos. Após pesquisas e coleta de informações, verificou-se que em relação ao item I, o sistema operacional Android atende melhor os requisitos, pois ao contrário do IOS em que existe mensalidade, o Android possui um sistema de pagamento único para tornar o indivíduo desenvolvedor e possibilitar o *upload* de seu aplicativo à loja virtual.

No Brasil, não há necessidade de procurar muito para constatar que os preços praticados por celulares com sistema operacional IOS são muito mais elevados que os com sistema operacional Android, outro fator muito importante a ser levado em

consideração, e que está presente no item II. Ao comparar os preços de celulares Android e IOS, fica claro que tanto para aquisição quanto uma possível distribuição em etapas posteriores do projeto, é mais interessante adquirir *smartphones* com sistema Android pela diferença de preço.

Em relação ao item III, o sistema Android acabou por ser também o mais atrativo, visto que se trata de um sistema aberto, muito mais acessível ao público e aos próprios desenvolvedores de aplicações. E que, de mesmo modo, possui sua principal plataforma de desenvolvimento (Android Studio) disponível de maneira gratuita. É exatamente pelo fato de ser uma plataforma gratuita em conjunto com preços acessíveis de desenvolvimento, que caso haja necessidade de algum tipo de informação relativa a esses programas, processos e aplicações, é possível encontrar diversas fontes que auxiliam o individuo a solucionar seus problemas e tirar suas dúvidas.

Como dito na explicação do item II, pelo fato de dispositivos com sistema Android serem mais acessíveis economicamente, baseando-se no cenário nacional atual, é natural que seu uso seja popular entre as pessoas (item IV). Então, avaliando a possibilidade de distribuir a aplicação em sistemas baseados em Android, pela familiaridade dos usuários com a plataforma, não seria difícil uma adaptação.

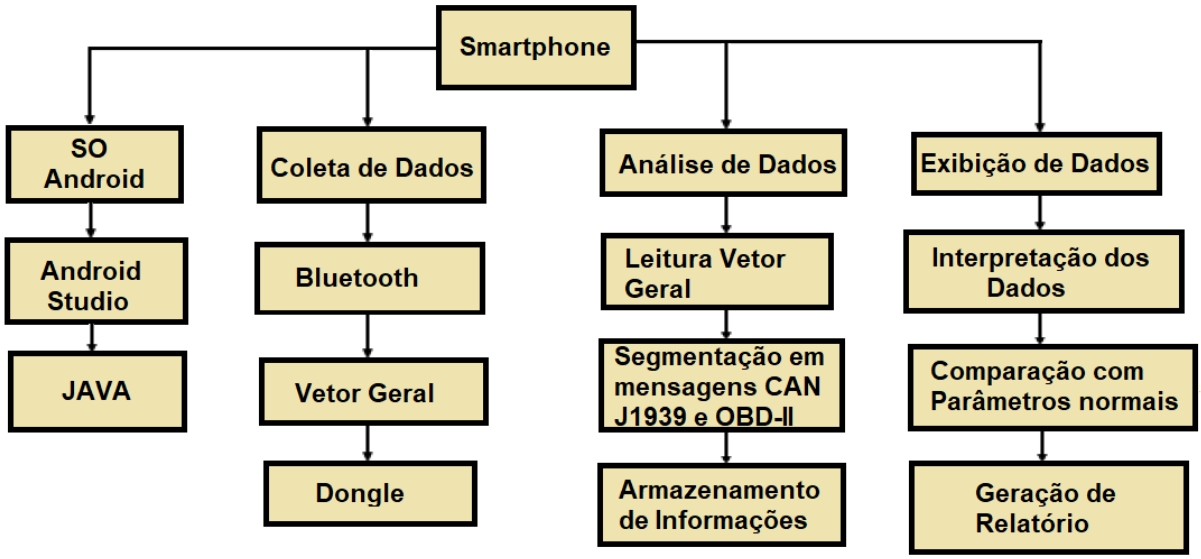
A linguagem de programação é um fator essencial e de suma importância para toda a aplicação, pois é a partir dela que o *software* será desenvolvido. Portanto, ao levar em consideração o uso de sistemas Android, a linguagem de programação que é comumente utilizada com essa plataforma é a linguagem JAVA. Então, atendendo os itens V e VI, essa linguagem torna-se viável pois além de ser compatível com o sistema operacional escolhido, também tem certa familiaridade com o desenvolvedor (autor), visto que há uma matéria que contempla essa linguagem no curso.

Por fim, o último item a ser levado em consideração (VII), necessitou de pesquisas dentro da empresa por soluções que envolviam aplicativos e foi constatado que também era um fator atendido, tendo como base que já foram desenvolvidas diversas soluções com aplicações distintas nesse sistema operacional no ambiente da empresa.

Vale destacar ainda que, portanto, o sistema operacional optado foi o Android, a

plataforma de desenvolvimento o Android Studio e a linguagem de programação JAVA. Então o celular, portador da solução e aplicação do *software* neste projeto, exercerá funções fundamentais, e para melhor ilustrar quais seus papeis no sistema, a Figura 3.5 foi elaborada.

Figura 3.5: Funcionalidades do *Smartphone* no Sistema



Fonte: O Autor (2018).

CAPÍTULO 4

IMPLEMENTAÇÃO

4.1 Layout

No inicio da concepção das ideias para o projeto, após discussões com a PRF objetivando compreender melhor o escopo, foi elaborado uma idealização do *layout* para tornar-se a base do desenvolvimento do projeto. É possível observar esses modelos no Apêndice B.

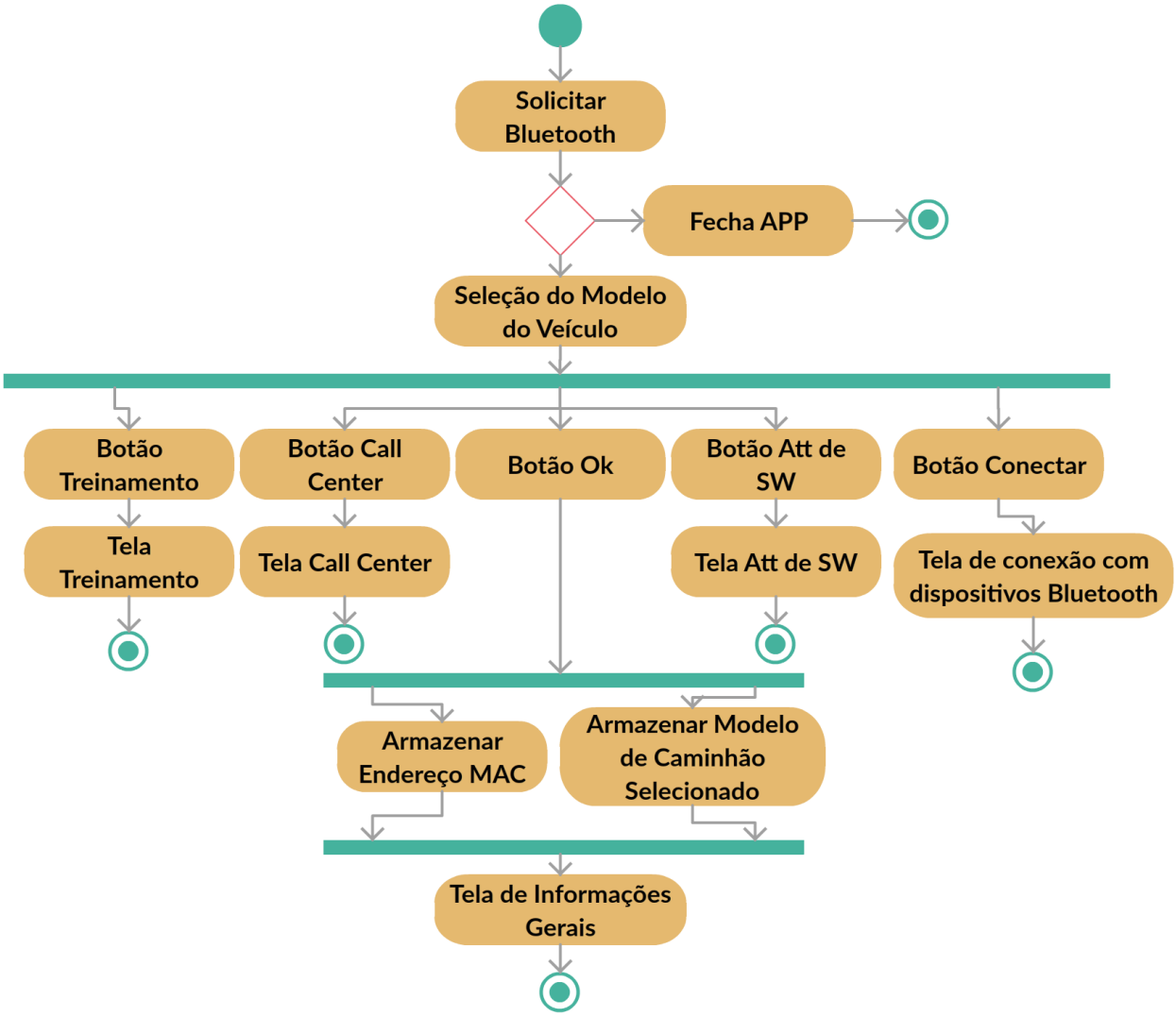
Após o estudo sobre a programação envolvida, procurou-se desenvolver as telas do aplicativo da maneira mais intuitiva possível e que fosse condizente com a proposta do projeto, aprimorando alguns aspectos de cada *layout* para que ficassem melhor distribuídos pelo espaço presente nas telas e aprimorando termos e indicativos fornecidos pelos clientes para tornar a experiência de uso amigável.

4.1.1 Tela Principal

Na tela inicial, Figura C.2, constam diversas opções de interação para o usuário, o intuito desta interface é o de permitir que o usuário possa acessar através dos botões o *call center* e o vídeo de treinamento caso não saiba como utilizar alguma parte do aplicativo ou esteja com alguma dúvida.

Também é possível verificar eventuais atualizações de *software*, conectar-se a algum dispositivo *Bluetooth* por perto e também selecionar o modelo do veículo que estiver analisando. Para a melhor compreensão do funcionamento, o diagrama de fluxo da classe foi feito e pode ser observado na Figura 4.1.

Figura 4.1: Diagrama da classe *MainActivity*.



Fonte: O Autor (2018).

Atrelada à tela principal, na Figura C.2 estão às classes *MainActivity* e *MainActivityFragment*, suas codificações podem ser vistas nos Apêndices J e K. Essas classes começam inicializando todas as ferramentas que serão usadas para lógicas e interações.

Dentre essas ferramentas, encontram-se os botões, a *String* para armazenar o valor do endereço MAC do dispositivo *Bluetooth* o qual estabelece conexão com o app, os adaptadores e *sockets* para utilizar o *Bluetooth*, uma variável booleana para definir se existe conexão ou não, e por fim algumas flags que serão usadas no decorrer da classe.

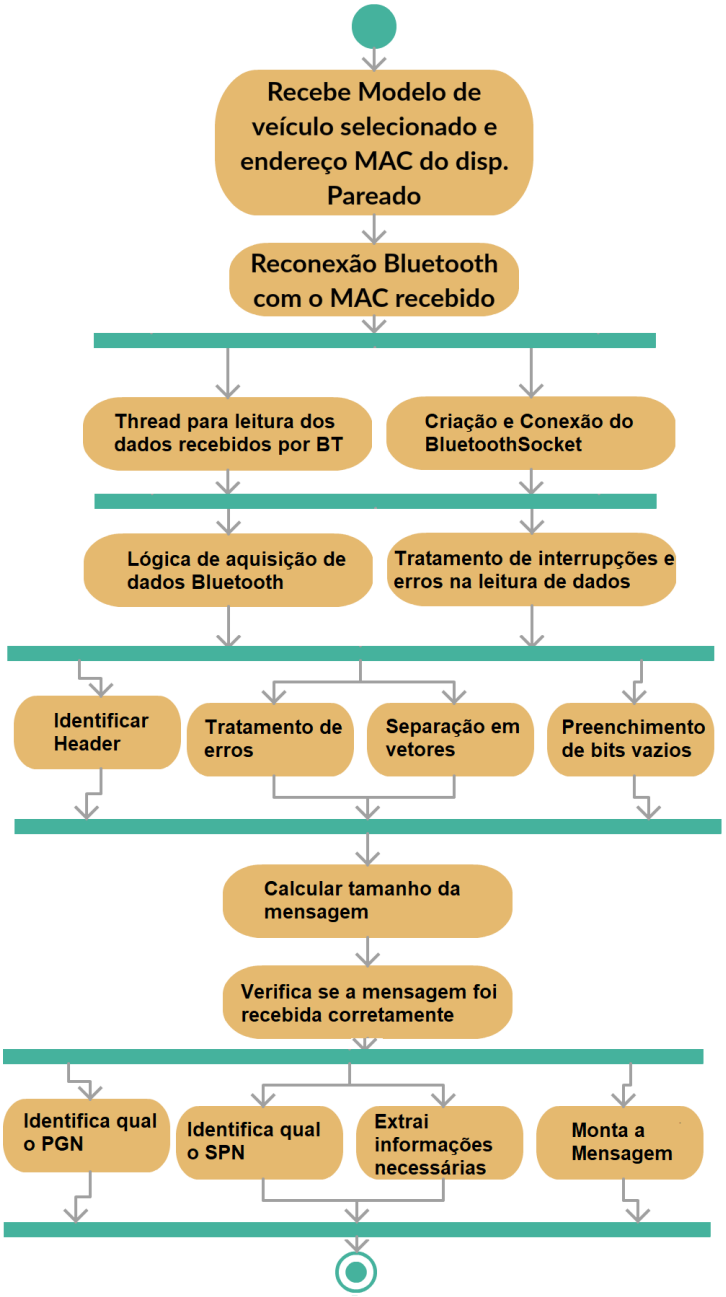
Após essas inicializações, a classe atrela a sua lógica à tela de início, inicia o *spinner*, que é o objeto que seleciona o modelo dos caminhões, relacionando a ele um *arraylist* dos caminhões que serão contemplados no app, solicita permissão do uso do *Bluetooth* do celular, e caso o usuário negue ele fecha o aplicativo. E por último, define as lógicas dos botões para mudar de tela dependendo do botão que for pressionado. Se o usuário pressionar o botão conectar a classe abre a lista de dispositivos que já estão pareados e também a de dispositivos disponíveis para pareamento. Se pressionar o botão do *call center*, mudará para a tela do *call center*, se pressionar o botão de atualização de *software*, mudará para a tela de atualização de *software*, se pressionar o botão de treinamento, mudará para a tela de treinamento e se pressionar o botão o*ok*, armazenará o endereço mac do dispositivo ao qual foi conectado e o caminhão que foi escolhido e levará essas informações para a próxima tela.

4.1.2 Tela de Informações Gerais

Após conectar-se com o dispositivo *Bluetooth* e selecionar o modelo do veículo que se deseja analisar, o aplicativo recebe o modelo que foi selecionado e indica ao usuário qual o protocólo de comunicação que aquele modelo de caminhão utiliza, também lê as informações do barramento CAN sobre o tempo da MIL acesa, o número VIN e de série do motor e os DTCs, tratando-os dados para exibição para o usuário. Os DTCs serão removidos na versão final do aplicativo, pois, no momento só foram inseridos na etapa atual com fins exclusivos de *debug* e verificação de leitura. A tela em questão pode ser observada na Figura C.2.

Nesta etapa, o foco foi no tratamento e aquisição correta dos dados vindos do barramento do veículo. Esses dados são recebidos e processados na tela de informações gerais, que pode ser vista na Figura C.2, também conhecida como a classe *VeriffActivity*, sua codificação nova e alterada pode ser vista no Apêndice P. Para a melhor compreensão do funcionamento e das mudanças realizadas na lógica anterior, o diagrama de fluxo da classe foi feito e pode ser observado na Figura 4.2.

Figura 4.2: Diagrama da classe *VeriffActivity*.



Fonte: O Autor (2018).

A classe começa inicializando todas as ferramentas que serão usadas para lógicas e interações, o botão para prosseguir para a próxima tela, uma flag utilizada para selecionar o protocólo de acordo com o modelo de caminhão escolhido na tela principal, as strings dos protocólos, do endereço MAC do dispositivo conectado, do número de série do motor, do tempo da MIL, do número VIN, o *socket Bluetooth*, a string

que receberá o caminhão selecionado anteriormente e o *hashmap* que exibirá as informações para o usuário. Após todas as inicializações necessárias, uma instância é executada para se comunicar com a instância da tela passada e pegar qual o modelo de caminhão o usuário selecionou e também o endereço MAC do dispositivo ao qual o app está conectado. Em seguida são atribuidos valores de *flags* para cada um dos modelos de caminhões disponíveis, então o método *initTextView* é chamado e dependendo do modelo é exibido um protocolo de comunicação correspondente, as informações referentes ao tempo da MIL e aos números VIN e de série foram pré-definidos no inicio da classe e serão exibidos. Como o usuário se conectou a um dispositivo *Bluetooth* ele reestabelece a conexão para garantir funcionamento.

Com a conexão estabelecida o *BluetoothSocket* é iniciado e uma *Thread* é executada para poder receber todos os dados sendo transmitidos por *Bluetooth*. Com uma lógica para tratamento de interrupções e erros de leitura para que se houver uma interrupção durante a aquisição de dados, o aplicativo não trave e também se houver algum erro de leitura o aplicativo possa contornar esse erro iniciando uma nova leitura. Para a aquisição dos dados, é inicializado o *InputStream* que recebe todas as informações e vai armazenando em uma variável que será lida e tratada posteriormente.

Após isso é chamado um método para analisar a mensagem e dividí-la em vetores para separar as mensagens que estão trafegando no barramento. Esse método começa identificando o *header* dos dados para determinar sempre onde se encontra o início de uma mensagem, depois cada posição posterior até o próximo header é selecionada e concatenada ao lado do *header* para formar um vetor equivalente à uma mensagem CAN. Em meio a esse preenchimento são executadas algumas lógicas para verificar se o byte foi preenchido corretamente e se a mensagem está sendo montada da maneira correta.

Depois que as mensagens são montadas, todas elas entram em um método chamado *processPacket* que pega os pacotes de informações, lê cada uma das mensagens CAN montadas e extrai as informações desejadas. Neste método os

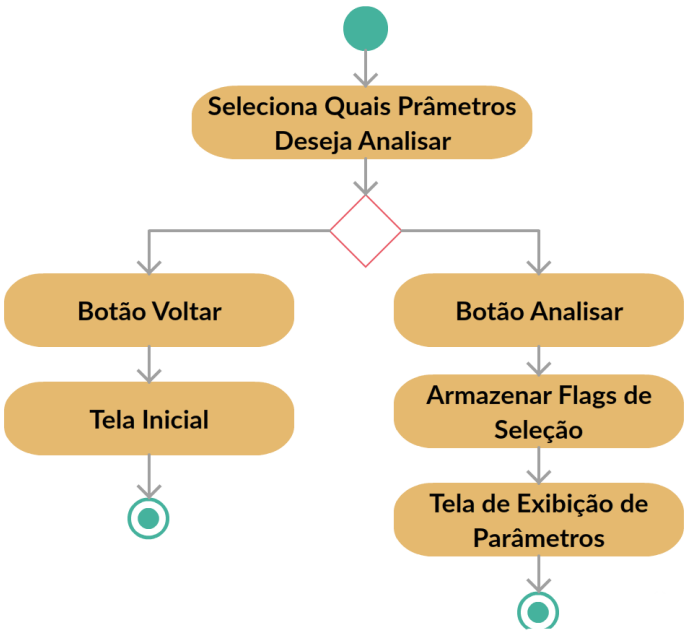
conceitos referentes ao protocolo CAN e J1939 são utilizados, primeiramente o header da mensagem é identificado para que a partir dele seja possível mapear onde está cada informação do pacote. Depois os bytes da mensagens referentes ao PGN são lidos e sempre que um dos valores de PGN desejados for lido, entra-se em um loop onde é possível fazer a análise, leitura e extração das próprias mensagens presentes dentro dos SPNs, que são bytes dentro de cada um dos PGNs. Vale lembrar que no código do Apêndice P os valores de SPN e PGN foram censurados pois são informações confidênciais.

Então após ler cada um dos SPNs e PGNs referentes às mensagens selecionadas (tanque vazio, nível de emissões, falha elétrica, etc...), seus valores respectivos são armazenados dentro de variáveis que posteriormente serão analizadas na *ShowActivity* para verificar se estão ou não dentro dos padrões e exibir para o usuário esse diagnóstico.

4.1.3 Tela de Escolha de Parâmetros

A próxima tela pela qual o usuário passará é a tela de escolha dos parâmetros a serem analisados. A Figura C.3 mostra como é a interface desta etapa, onde o usuário poderá escolher quais parâmetros disponíveis deseja analisar em sua diagnose atual. Se deseja analisar se o tanque de ARLA32 está vazio, se o nível de emissões está acima do normal, se o nível de emissões está em um nível muito grave, se há algum tipo de falha de comunicação no sistema, se há algum tipo de falha elétrica e se há algum valor lido que seja inconsistente. Para a melhor compreensão do funcionamento, o diagrama de fluxo da classe foi feito e pode ser observado na Figura 4.3.

Figura 4.3: Diagrama da classe *ChooseActivity*.



Fonte: O Autor (2018).

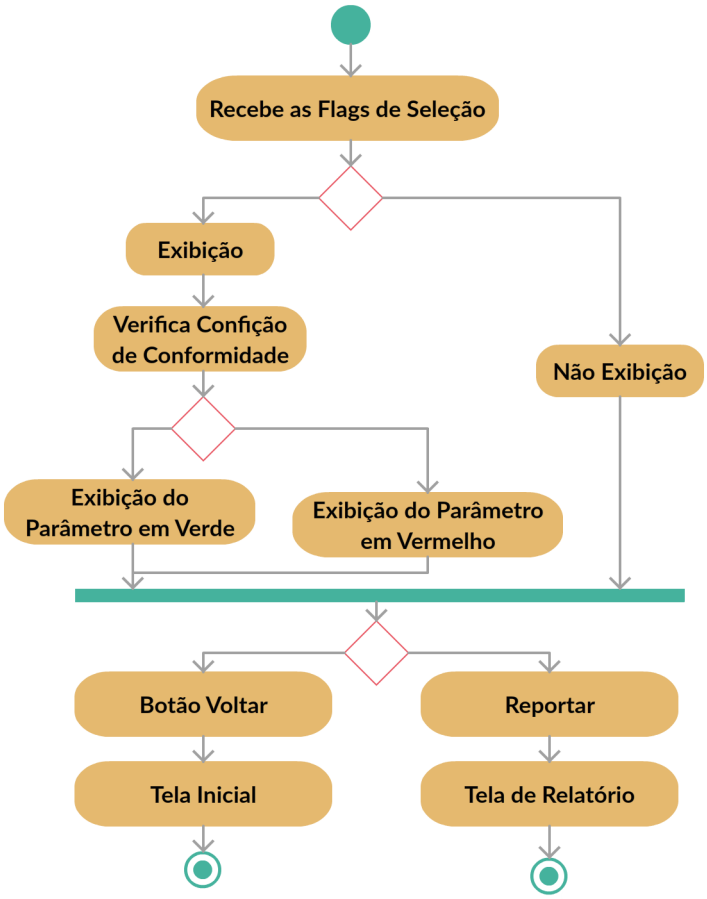
Atrelada à tela de escolha de parâmetros, Figura C.3 está a classe *ChooseActivity*, sua codificação pode ser vista no Apêndice F. Essa classe começa inicializando todas as ferramentas que serão usadas para lógicas e interações, o botão para prosseguir para a próxima tela, o botão para voltar para a tela inicial e as flags utilizadas para indicar se foram ou não selecionados cada um dos itens de análise. Após essas inicializações, é feito uma rotina que verifica se cada um dos itens está selecionado ou não, pois o intuito nesta tela é deixar como opção do usuário selecionar o que ele deseja analisar, e então atribui o valor 1 à flag caso esteja selecionado e 0 se não estiver. Em seguida o usuário tem a opção de voltar à tela inicial ou ir para a próxima. Se ele optar por continuar é inicializada uma instância que armazenará os valores dessas flags para enviar como pacote para a próxima classe.

4.1.4 Tela de Análise de Parâmetros

Após selecionar quais paramêtros se deseja analisar no veículo em questão, a próxima interface a ser exibida é a da Figura C.5. Esta tela mostra todos os parâmetros

selecionados, mudando suas respectivas cores de acordo com o comportamento padrão ou não padrão. Para a melhor compreensão do funcionamento, o diagrama de fluxo da classe foi feito e pode ser observado na Figura 4.4.

Figura 4.4: Diagrama da classe *ShowActivity*.



Fonte: O Autor (2018).

Atrelada à tela de exibição de parâmetros, Figura C.5 está a classe *ShowActivity*, sua codificação pode ser vista no Apêndice L. Essa classe começa inicializando todas as ferramentas que serão usadas para lógicas e interações, o botão para prosseguir para a próxima tela, o botão para voltar para a tela inicial, as variáveis booleanas para indicar simbólicamente se houve falha elétrica, de comunicação ou valor inconsistente, o valor do nível do tanque de ARLA32 e os valores de emissões em partes por milhão.

Após essas inicializações, é feito uma instância que irá adquirir os valores das flags recebidas pelo pacote enviado pela classe *ChooseActivity*, em seguida os *textviews* são inicializados e atrelados aos seus *ids* respectivos no *layout*. Em seguida, a classe

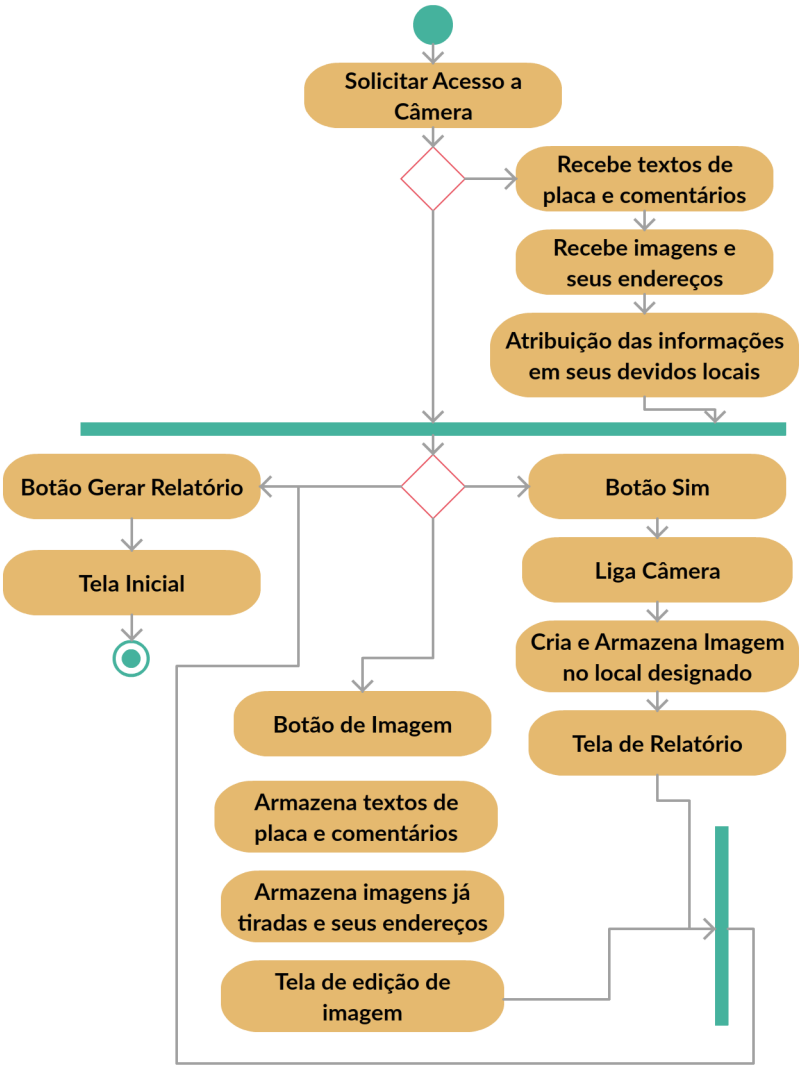
verifica para cada um dos parâmetros a serem analizados, se a flag de seleção na tela anterior estava em 1 (selecionado) ou em 0 (não selecionado), pois se o valor for 1 aquele parâmetro será exibido e se for 0 será ocultado. Para a exibição, cada parâmetro tem sua lógica para verificar se será exibido em verde (dentro do esperado) ou vermelho/amarelo (fora do esperado), o vermelho foi selecionado para parâmetros que se forem detectados, indicam um caso certo de bula no sistema de pós-tratamento, enquanto os exibidos em amarelo se forem detectados podem ser em decorrência de uma burla. Os métodos para identificação de conformidade ou não dos parâmetros com os dados fornecidos pelo veículo serão implementados em etapas posteriores quando o dispositivo de comunicação for escolhido.

4.1.5 Tela de Relatório

Por fim, depois dos parâmetros terem sido análisados o usuário tem a opção de gerar um relatório com as principais etapas de todo o processo de diagnose. A Figura C.3 exemplifica como é a interface de preenchimento antes de gerar o relatório. Nesta etapa o usuário tem a opção de inserir a placa do veículo, eventuais comentários que julgar relevante e também tirar fotos, as quais podem ser armazenadas em no máximo grupos de 6. Portanto, quando o usuário preencher todas as partes que julgar importantes e pressionar o botão de gerar relatório as informações mais importantes, tais como as digitadas na interface atual, as informações exibidas na tela da Figura C.2 e as eventuais falhas exibidas na interface da Figura C.5, serão reunidas neste documento e armazenadas no cartão SD (*Secure Digital*).

Atrelada à tela de relatório, Figura C.3 está a classe *InformationActivity*, sua codificação pode ser vista no Apêndice I. Para a melhor compreensão do funcionamento, o diagrama de fluxo da classe foi feito e pode ser observado na Figura 4.5.

Figura 4.5: Diagrama da classe *InformationActivity*.



Fonte: O Autor (2018).

Essa classe começa inicializando todas as ferramentas que serão usadas para lógicas e interações, o botão para prosseguir para a realização do relatório e voltar para a tela inicial, o botão para acessar a câmera e tirar fotos, uma *string* para armazenar posteriormente o local da imagem, um vetor de imagens para armazenar as 6 imagens disponíveis no app, um vetor de *strings* para armazenar os endereços das imagens e duas *strings* para armazenar o que foi escrito na placa do veículo e nos comentários. Após essas inicializações, a classe solicita para o usuário a permissão de acessar a câmera do dispositivo, o botão para finalizar o processo é declarado e se pressionado volta para a tela inicial. Em seguida os campos para inserir texto, tanto da parte da

placa do veículo quanto da parte de comentários são declarados, o vetor de 6 imagens têm cada uma de suas posições atreladas aos locais onde residem os ícones das máquinas fotográficas na tela, para que depois que as fotos forem tiradas elas sejam armazenadas em cada um deles.

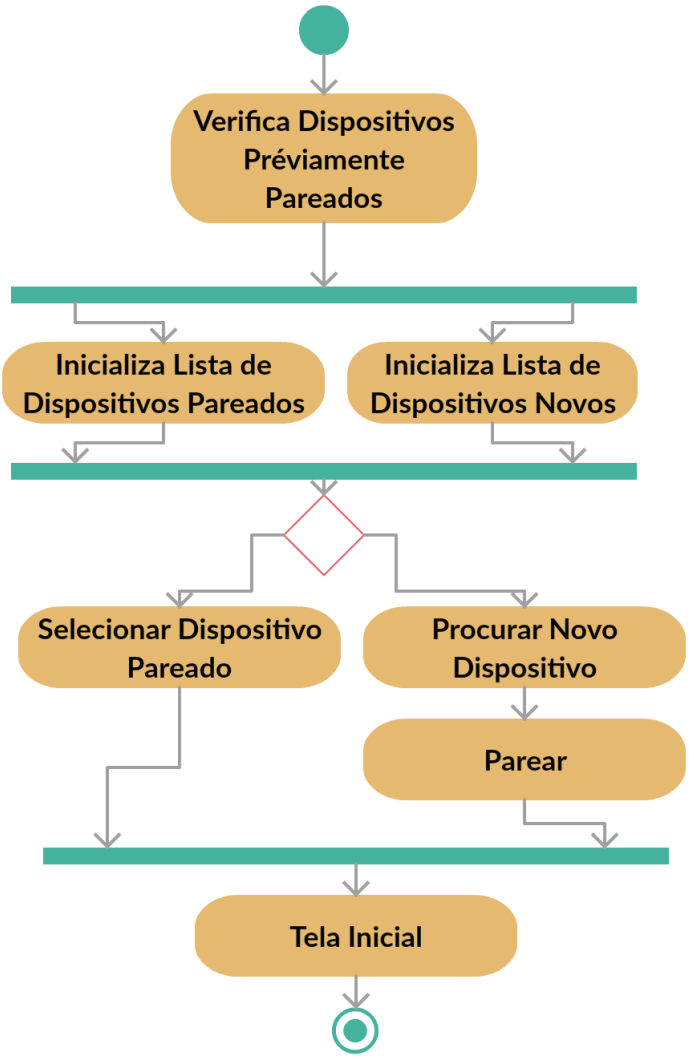
Uma instância é criada para quando retornar da tela de edição da imagem, receba os endereços das imagens já tiradas e os textos já digitados nos campos respectivos, reatribuindo as imagns que não foram alteradas para seus locais de origem. O botão de acessar a câmera é declarado e quando pressionado retorna a imagem tirada para a primeira posição dos vetores de imagens que estiver sem nada. No método criarImagem são definidos alguns padrões de nomenclatura para as imagens tiradas e a seguir é criada a imagem .jpg quando tirada com a câmera. O método setImageListener armazena e transfere na hora de mudar de tela para a que está atrelada à classe *ImageActivity* todas as imagens que já foram tiradas e textos que já foram escritos, bem como os endereços dessas imagens. Os métodos para a criação do PDF com o relatório contendo as falhas, imagens e informações relevantes serão implementados em etapas posteriores quando o aplicativo já estiver comunicando com o caminhão e recolhendo os dados.

4.1.6 Outras telas

As interfaces restantes, referentes ao *call center*, video de treinamento, tela de carregamento e de atualização de *software* estão sendo contempladas no Apêndice C, nas Figuras C.1 e C.4.

Atrelada às telas de seleção de dispositivos *Bluetooth* está a classe *DeviceListActivity*, sua codificação pode ser vista no Apêndice G. Essa classe começa inicializando todas as ferramentas que serão usadas para lógicas e interações, o adaptador *Bluetooth* do dispositivo, e dois *arraylists*, um para os dispositivos novos e outro para os dispositivos préviamente pareados com o aparelho celular. Para a melhor compreensão do funcionamento, o diagrama de fluxo da classe foi feito e pode ser observado na Figura 4.6.

Figura 4.6: Diagrama da classe *DeviceListActivity*.

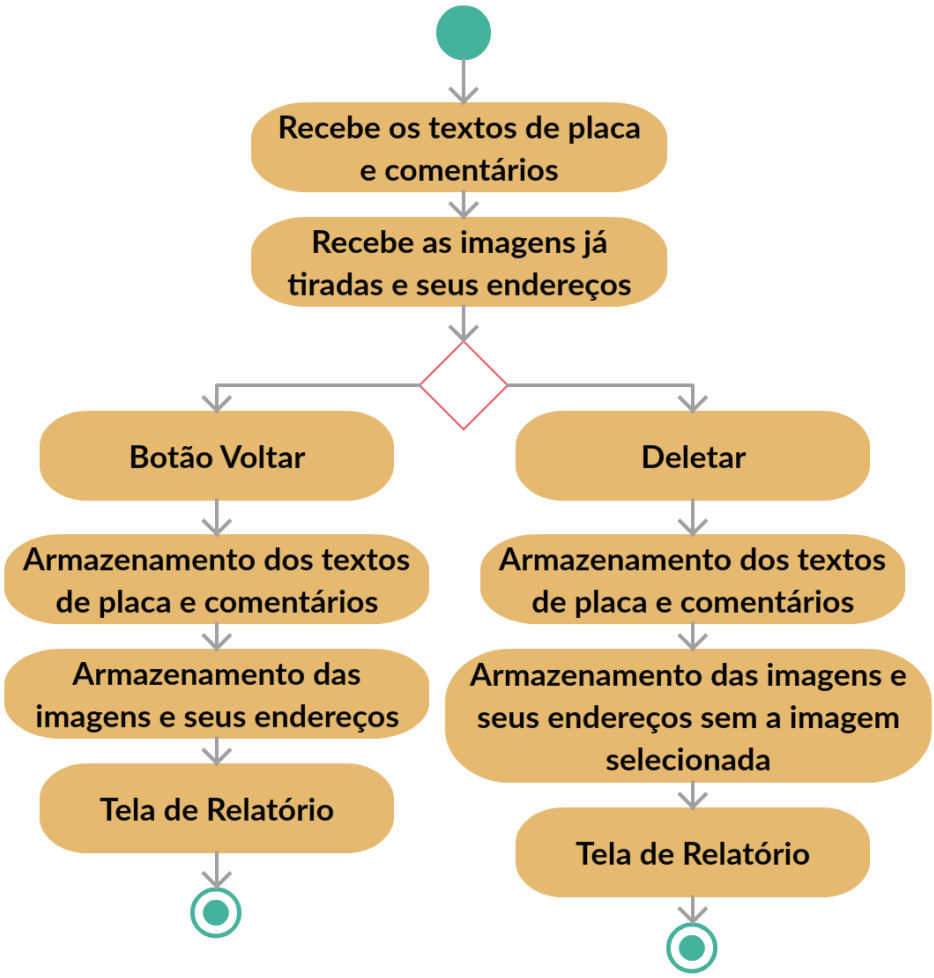


Fonte: O Autor (2018).

Após essas inicializações, a classe configura a tela e inicializa o botão que permite ao usuário procurar novos dispositivos. Em seguida os *arrayadapters* são inicializados e a lista de dispositivos pareados é encontrada e configurada, assim como a de novos dispositivos. Em seguida é feito a classe faz com que o celular seja registrado para receber conexões quando um novo dispositivo for descoberto e os adiciona à lista. O método *doDiscovery* presente na classe é implementado para comutar os títulos da lista e exibir o que está ocorrendo para o usuário, se está procurando por dispositivos ou não, caso seja descoberto algum dispositivo novo por perto ele também adiciona-o à lista de dispositivos novos e se não tiver ninguém exibe a mensagem ”Nenhum

Dispositivo Encontrado". Para a melhor compreensão do funcionamento, o diagrama de fluxo da classe foi feito e pode ser observado na Figura 4.7.

Figura 4.7: Diagrama da classe *ImageActivity*.



Fonte: O Autor (2018).

Atrelada à tela de edição de imagem, está a classe *ImageActivity*, sua codificação pode ser vista no Apêndice H. Essa classe começa inicializando todas as ferramentas que serão usadas para lógicas e interações, o botão para voltar para a tela de relatório, o botão para deletar a imagem selecionada e o *imageview*.

Após essas inicializações, é inicializada a instância que recebe o que foi digitado na tela anterior na placa do veículo e nos comentários, e também as imagens que já foram tiradas e seus endereços. Em seguida o usuário tem duas opções, pressionar o botão voltar para retornar para a tela de relatório e continuar realizando suas avalizções ou

pressionar o botão deletar para apagar a imagem selecionada e voltar para a tela de relatório. Se o botão voltar for selecionado, a instância retorna para a classe *InformationActivity* os textos digitados anteriormente na própria classe *InformationActivity* e as imagens que já foram tiradas, já se o usuário selecionar o botão deletar, a instância retorna para a classe *InformationActivity* os textos que já foram digitados e as imagens que já foram tiradas sem aquela que acabou de ser apagada.

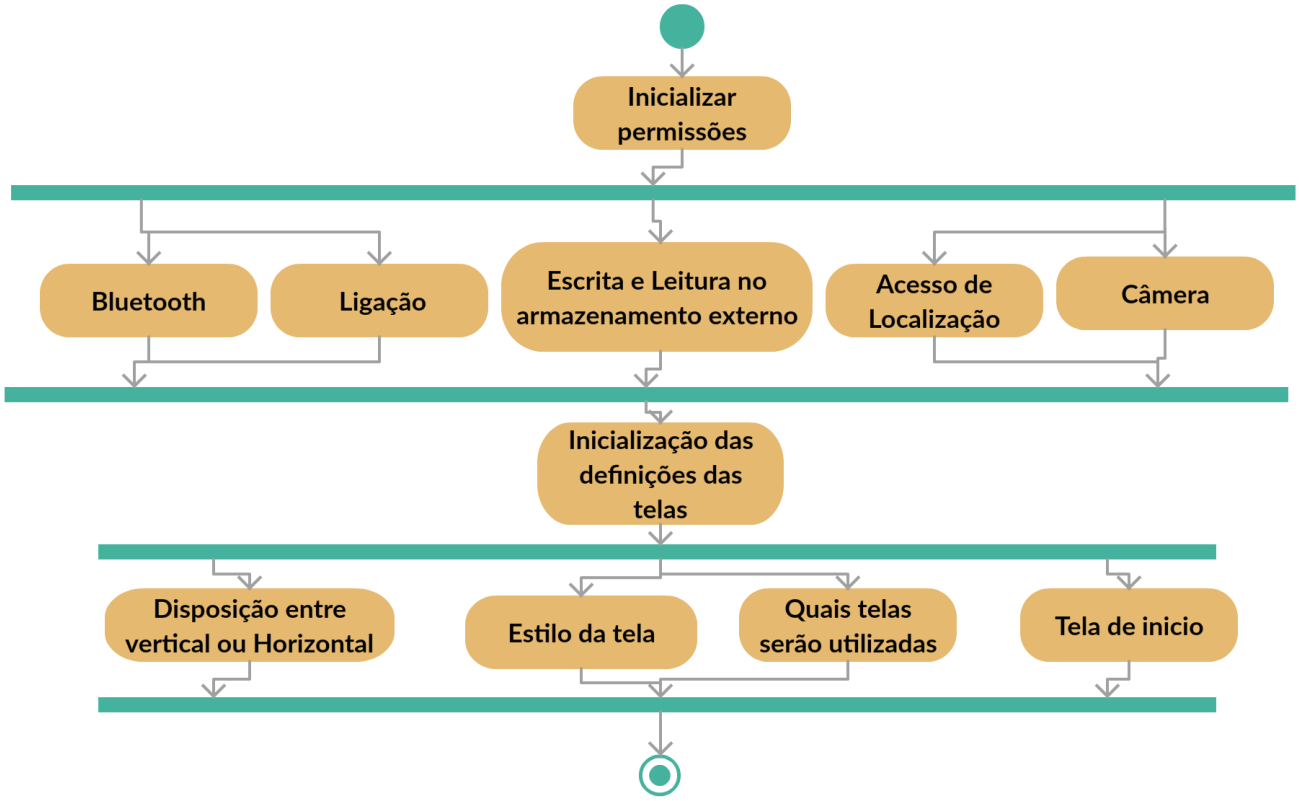
4.2 Lógica de Programação

Com os *layouts* prontos, a equipe pode ter uma visão completa de como ficaria o aplicativo em termos de aparência e também quais partes teriam que possuir alguma lógica por trás para que se comportassem como o desejado. Depois desta análise a equipe procurou desenvolver as lógicas condizentes com a proposta do projeto e a distribuição dos itens na tela, aprimorando alguns aspectos para que ficassem com uma velocidade de resposta otimizada. Mais detalhes sobre a implementação das lógicas em cada *layout* e os resultados obtidos, bem como os diagramas para melhor vizualização podem ser encontrados no Capítulo 4.

4.3 Lógica - Manifesto

A codificação do manifesto utilizada na aplicação deste projeto pode ser vista no Apêndice D. A classe se inicia informando que haverão solicitações de permissão de uso no decorrer da utilização do aplitavivo, para ligação do celular, escrever e ler no armazenamento externo do aplicativo, uso do *Bluetooth*, acesso à localização atual e a câmera do dispositivo. Após esta etapa, são definidas as informações referentes ao ícone do aplicativo no aparelho, tais como o nome e a imagem do ícone. Por fim, são informadas informações relevantes sobre as telas que o aplicatico irá utilizar, qual tela será a tela de inicio, se elas serão exibidas com flexibilidade ou se serão travada em exibição vertical ou horizontal. Para a melhor compreensão do funcionamento, o diagrama de fluxo da classe foi feito e pode ser observado na Figura 4.8.

Figura 4.8: Diagrama da classe *Manifest*.

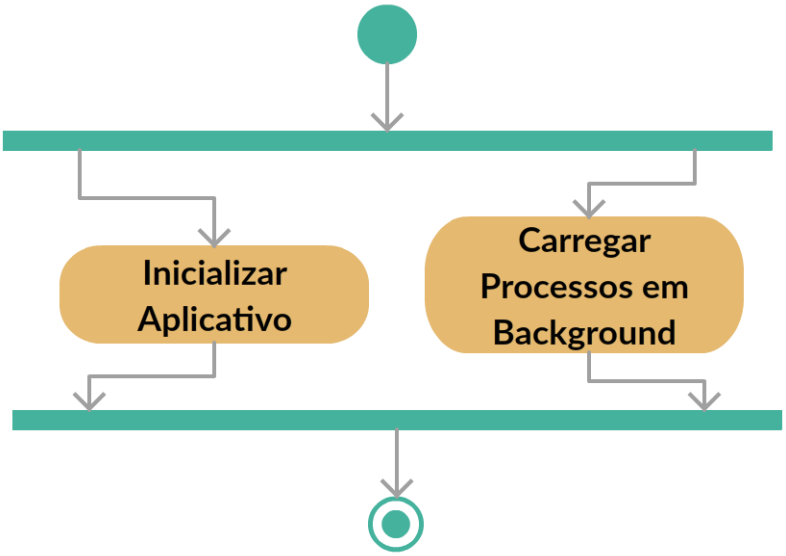


Fonte: O Autor (2018).

4.4 Lógica - Tela de Carregamento

A primeira tela a ser exibida no aplicativo é a tela de carregamento, Figura C.4. A codificação relacionada a ela é a feita na classe *SplashActiviy* e pode ser vista no Apêndice M. Esta classe não pode sofrer interação do usuário e permanece exibida apenas por alguns segundos, ela é responsável por inicializar um carregamento prévio antes de iniciar própriamente o aplicativo para que enquanto isso, se existirem, os processos que demandam mais tempo de carregamento possam ser carregados neste meio tempo. Assim sendo, enquanto essa tela está sendo exibida o aplicativo já executa os procedimentos mais demorados, para que assim seja possível uma otimização de tempo de processamento e uma da uma ”ilusão”para o usuário de que o aplicativo é bem eficiente e veloz. Para a melhor compreensão do funcionamento, o diagrama de fluxo da classe foi feito e pode ser observado na Figura 4.9.

Figura 4.9: Diagrama da classe *SplashActivity*.

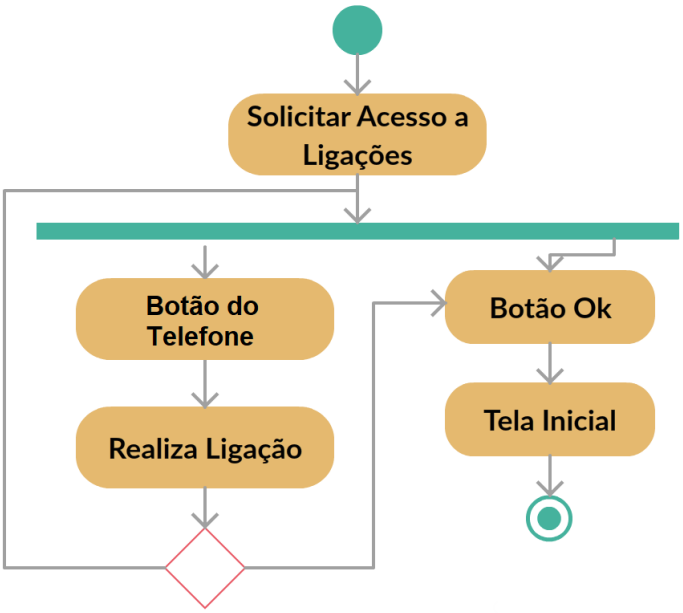


Fonte: O Autor (2018).

4.5 Lógica - Tela de Call Center

Atrelada à tela do *call center* está a Figura C.1, presente no Apêndice C e que representa a classe *CallActivity*, sua codificação pode ser vista no Apêndice E. Essa classe começa inicializando todas as ferramentas que serão usadas para lógicas e interações, os botões, a flag que será usada no decorrer da classe e a instância utilizada na hora em que os botões forem pressionados. Após essas inicializações, a classe solicita acesso ao sistema de ligações do celular, após isso entra e em um dos dois métodos assim que houver alguma interação com o usuário. Se o usuário pressionar o botão de ligação, a classe inicia uma ligação pelo celular para o número pré estabelecido pelo programador e se pressionar o botão *ok* ele retorna para a tela principal. Para a melhor compreensão do funcionamento, o diagrama de fluxo da classe foi feito e pode ser observado na Figura 4.10.

Figura 4.10: Diagrama da classe *CallActivity*.



Fonte: O Autor (2018).

4.6 Lógica - Tela de Treinamento

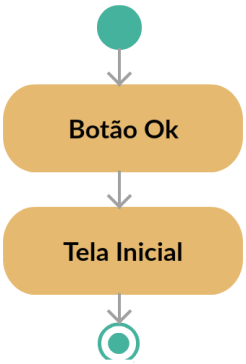
Atrelada à tela do treinamento está a Figura C.1, presente no Apêndice C e que representa a classe *TrainingActivity*, sua codificação pode ser vista no Apêndice N. Essa classe começa inicializando todas as ferramentas que serão usadas para lógicas e interações, nesse caso, o botão de retorno. Após essa inicialização, o usuário terá a opção de selecionar no botão *ok*, e então retornará para a tela principal. A implementação do video de instruções será realizada em etapas posteriores quando o aplicativo estiver finalizado.

4.7 Lógica - Tela de Atualização

Atrelada à tela de atualização de*software*, Figura C.4 está a classe *UpdateActivity*, sua codificação pode ser vista no Apêndice O. Essa classe começa inicializando todas as ferramentas que serão usadas para lógicas e interações, nesse caso, o botão de retorno. Após essa inicialização, o usuário terá a opção de selecionar no botão *ok*, e

então retornará para a tela principal. A implementação do processo de atualização de *software*, seja via nuvem ou *Google Play Store* será realizada em etapas posteriores quando o aplicativo estiver finalizado. Para a melhor compreensão do funcionamento, o diagrama de fluxo da classe foi feito e pode ser observado na Figura 4.11.

Figura 4.11: Diagrama da classe *UpdateActivity*.



Fonte: O Autor (2018).

4.8 Pesquisas sobre protocolos usados

Com as funcionalidades básicas de todas as telas do aplicativo funcionando e também com a validação de suas aplicações na última entrega, foi possível continuar com as pesquisas para analisar qual o protocolo deveria ser utilizado e decodificado na aplicação para abranger a maior gama de veículos possível. Após pesquisas foi descoberto que todos os caminhões que circulam nas estradas do país são obrigados a manterem as mensagens nos barramentos de acordo com o protocolo J1939 citado anteriormente na fundamentação teórica. Portanto, após o aprofundamento na estruturação das mensagens CAN e J1939 a equipe está apta para nas próximas entregas desenvolver o código para recepção dessas mensagens através do aplicativo e desmembramento do vetor de informações para análise e verificação de conformidade com os parâmetros adotados no aplicativo.

CAPÍTULO 5

TESTES

Após todos os fundamentos teóricos terem sido estudados, o projeto ter sido estruturado e os códigos e sistema implementados. A etapa seguinte são os testes, feitos para validar todas as teorias, requisitos, estruturações e principalmente funcionalidades necessárias para obter um resultado satisfatório na aplicação do projeto.

É de suma importância que seja feito, no inicio, um planejamento de todos os testes a serem realizados. Assim, se torna possível o mapeamento de todas as funcionalidades que deverão ser validadas e torna todo esse processo mais prático e intuitivo, visto que se planeja a ordem e natureza dos testes de antemão, já prevendo uma sequencia coesa. Para o projeto tratado neste documento, os testes realizados foram os seguintes:

- I. Aplicação Dummy;
- II. Conexão Bluetooth;
- III. Leitura de Parâmetros;
- IV. Condições de Anormalidade;
- V. Relatório.

Todos estes 5 testes foram planejados com o objetivo de analisar e validar o funcionamento dos principais pontos de aplicação do sistema, desde a conexão com o veiculo até a leitura, interpretação e exibição dos dados para o usuário. O resultado dos testes podem ser observados na Figura 5.1, e seus detalhamentos serão tratados nas próximas seções.

Figura 5.1: Lista dos Testes Estabelecidos

Teste	Comentário	Resultado
Aplicação Dummy	Funcionalidade e presença de todos os Layouts Validação da estrutura	Satisfatório
Conexão Bluetooth	Conexão estabelecida com dispositivo dongle	Satisfatório
Leitura de Parâmetros	Aquisição dos parâmetros essenciais	Satisfatório
Condições de Anormalidade	Indicações de alerta baseadas em cada uma das leituras com valores fora do padrão	Satisfatório
Album de Fotos	Funcionalidade de armazenar e apagar fotos	Satisfatório
Relatório	Criação e estrutura do relatório	Regular

Fonte: O Autor (2018).

5.1 Aplicação Dummy

Para esta etapa, visando a validação da aplicação *dummy* do software, foi feito um teste:

- **TD1:** Aplicação Dummy do *software*
- **Objetivo:** Validar o funcionamento do aplicativo e todas as suas telas sem as lógicas funcionais (analise dos dados, exibição de diagnóstico, geração do relatório).
- **Resultado:** Satisfatório

Para este teste, realizou-se uma idealização dos *layouts* das telas do aplicativo que posteriormente foi apresentada para os usuários finais (PRF), essas idealizações podem ser observadas no Apêndice B. Após receber os *feedbacks*, verificou-se a necessidade da mudança nos termos utilizados para designar os parâmetros analisados e também a adição de uma nova tela que pudesse receber fotos, comentários e gerar um relatório.

Tendo as mudanças em mente, os *layouts* finais do Apêndice C foram codificados e novamente levados para a PRF para uma última seção de *user experience*, apenas com suas funções dos botões de passar de tela em funcionamento, visto que era para ser um protótipo *dummy*. Com base neste evento e nos comentários positivos, foi

concluído que os testes para o protótipo foram satisfatórios.

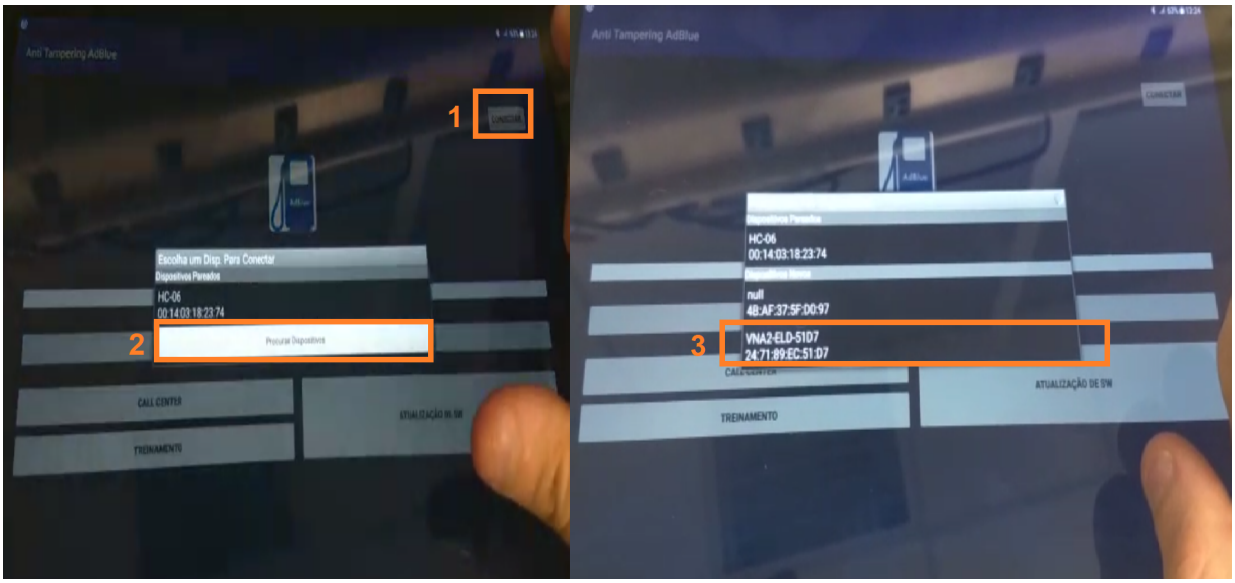
5.2 Conexão Bluetooth

Para esta etapa, visando a validação da comunicação *Bluetooth* entre *smartphone* e *dongle*, foram feitos dois testes:

- **TB1:** Detecção do *dongle*
- **Objetivo:** Validar o funcionamento da lista de dispositivos *Bluetooth* disponíveis para conexão.
- **Resultado:** Satisfatório
- **TB2:** Conexão *Bluetooth* pareada
- **Objetivo:** Validar o funcionamento e conexão correta do pareamento com o dispositivo selecionado da lista.
- **Resultado:** Satisfatório

Para o teste (TB1) foi utilizado o laboratório da empresa, onde todos os equipamentos necessários se faziam presentes, e o procedimento foi o seguinte: primeiro conetou-se o *dongle* à uma fonte de 5V para energizá-lo. Então, após as codificações necessárias, ligou-se o aplicativo em sua tela principal, clicando no botão conectar no canto superior direito e depois em 'Procurar Dispositivo' com o intuito de verificar se apareceria o *dongle* na lista dos dispositivos possíveis para pareamento. Este procedimento pode ser visto na Figura 5.2.

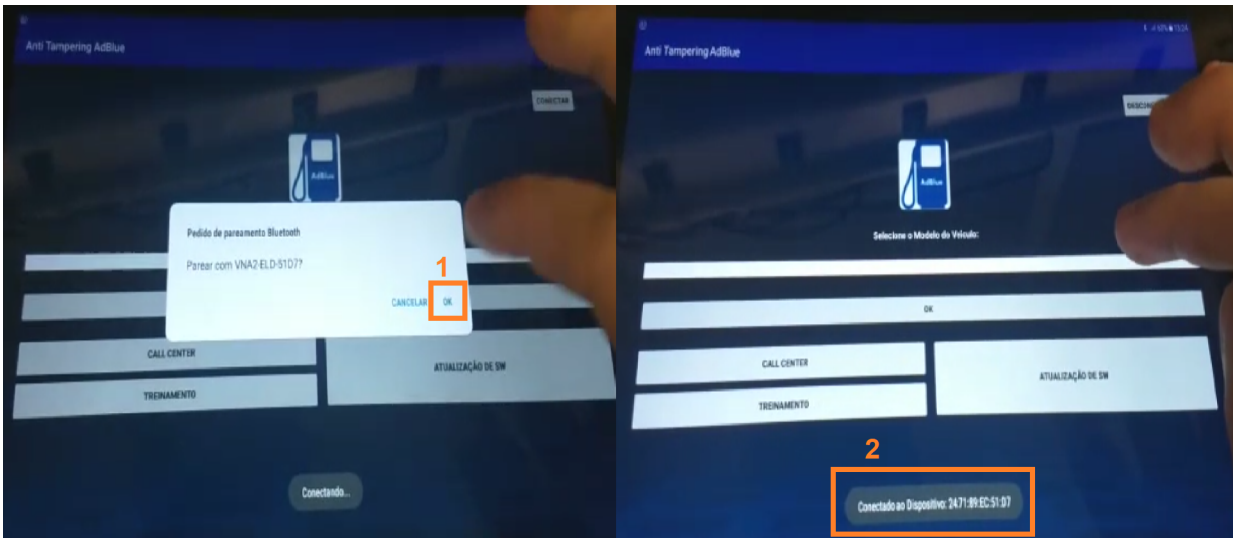
Figura 5.2: Teste *Bluetooth* TB1



Fonte: O Autor (2018).

Com a confirmação do funcionamento do teste (TB1), o próximo passo para executar o teste (TB2) foi clicar no *dongle* (VNA2-ELD-S1D7), que apareceu na lista e aguardar pela mensagem de pareamento. Quando a mensagem apareceu, o próximo passo foi clicar no 'Ok' e aguardar pelo aviso de confirmação do estabelecimento dessa conexão. Este procedimento pode ser visto na Figura 5.3.

Figura 5.3: Teste *Bluetooth* TB2



Fonte: O Autor (2018).

5.3 Leitura de Parâmetros

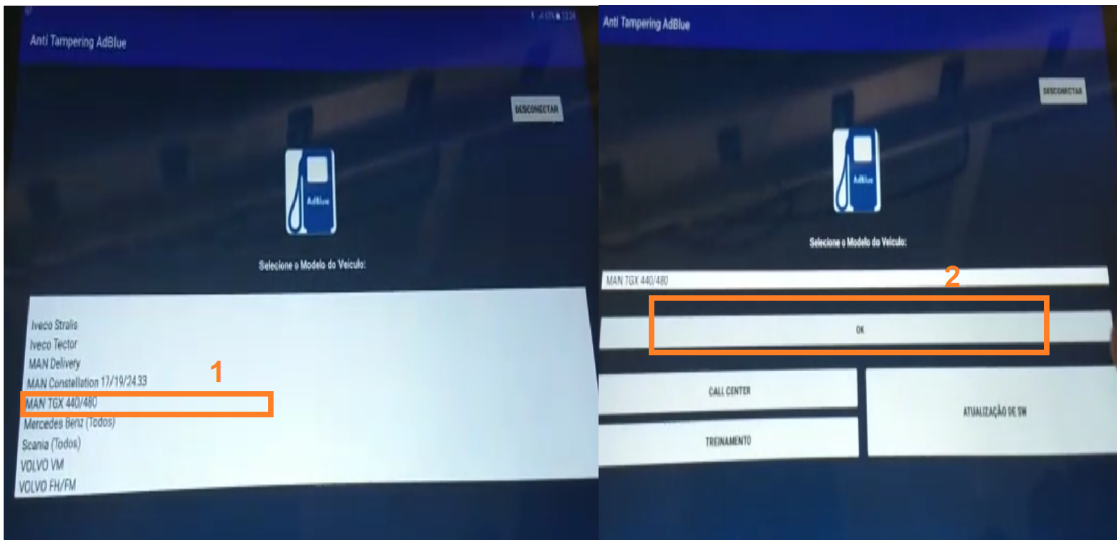
Para esta etapa, visando a validação da aquisição dos parâmetros entre *smartphone* e barramento CAN, foram feitos dois testes:

- **TP1:** Detecção de parâmetros gerais
- **Objetivo:** Validar a detecção e exibição dos parâmetros gerais da ECU.
- **Resultado:** Satisfatório
- **TP2:** Detecção de parâmetros específicos
- **Objetivo:** Validar a comunicação e detecção dos parâmetros específicos a serem analisados pelo *software*.
- **Resultado:** Satisfatório

Para o teste (TP1) foi utilizado o laboratório da empresa, onde todos os equipamentos necessários se faziam presentes, e o procedimento foi o seguinte: primeiro conetou-se o *dongle* à uma fonte de 5V e a ECU a uma fonte de 24 V para energizá-los, em seguida conectou-se a ECU a um trennadapter para que fosse possível ter acesso aos seus terminais, principalmente à alimentação e barramento CAN.

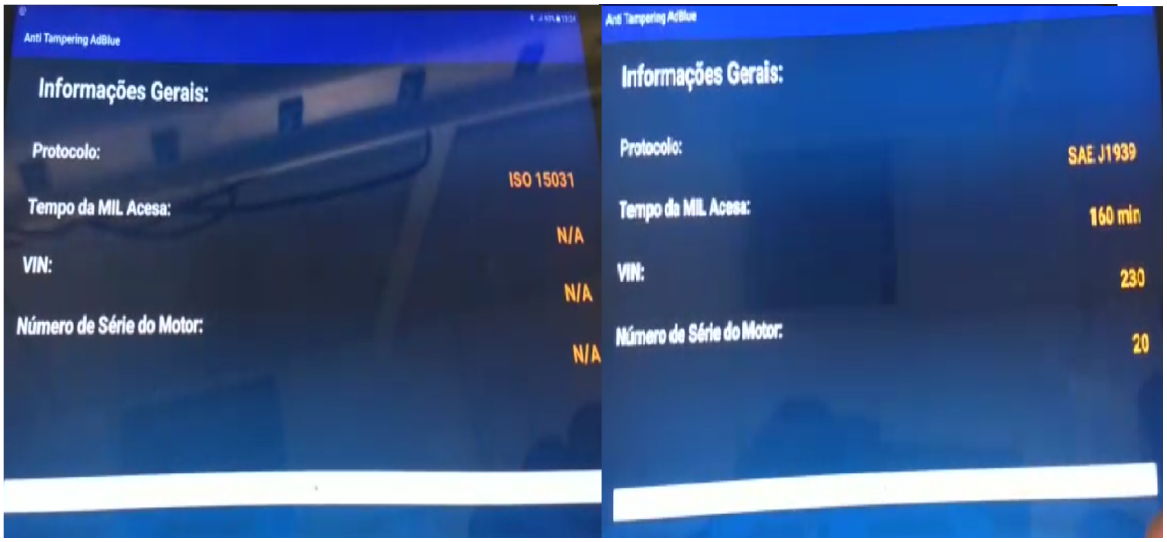
Em seguida conectou-se o barramento CAN ao *dongle* para simular a conexão direta ao veiculo e para que o dispositivo pudesse adquirir as informações passando no barramento daquela ECU. Então, após as codificações necessárias, ligou-se o aplicativo em sua tela principal, clicando no botão conectar no canto superior direito e depois no *dongle* na lista dos dispositivos possíveis para pareamento. Com a conexão estabelecida, selecionou-se o modelo do veiculo e o depois o botão 'Ok', procedimento exemplificado na Figura 5.4. Na tela posterior buscou-se verificar se os dados estavam preenchidos, o que indica uma conexão e troca de informações perfeitamente realizadas, visto que se não houvesse apareceriam dados indicando. Este procedimento pode ser visto na Figura 5.2.

Figura 5.4: Seleção de modelo, Teste TP1



Fonte: O Autor (2018).

Figura 5.5: Teste Parâmetros TP1

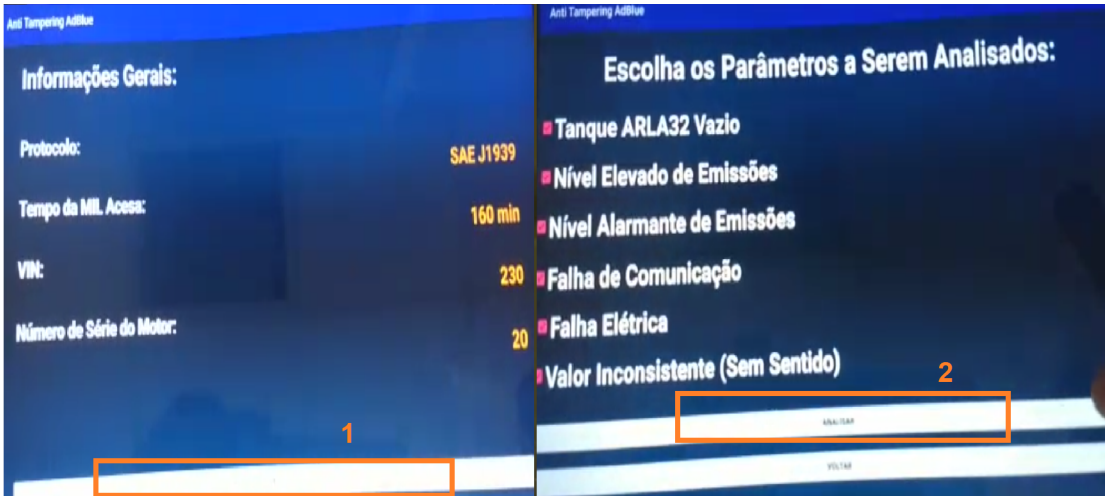


Fonte: O Autor (2018).

Para o teste (TP2), o procedimento foi clicar no ’Ok’ da tela de informações gerais, depois selecionar todos os parâmetros que se deseja analisar e pressionar o botão ’Analisar’, procedimento exemplificado pela Figura 5.6. Ao chegar na tela de exibição dos dados, espera-se encontrar todos aqueles parâmetros da tela anterior indicados de alguma cor. De acordo com a Figura 5.7, o teste foi satisfatório, visto que ao lado esquerdo é possível ver que sem conexão e comunicação os parâmetros ficam vazios

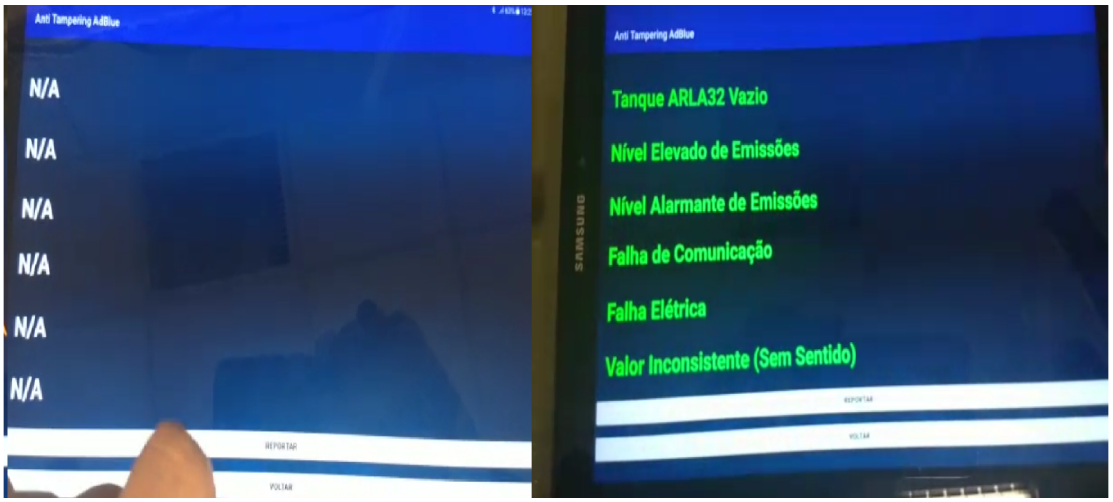
e com a conexão assegurada e comunicação executada é possível ler os dados e exibir os parâmetros já com o parecer.

Figura 5.6: Teste Parâmetros TP2 Pt.I



Fonte: O Autor (2018).

Figura 5.7: Teste Parâmetros TP2 Pt. II



Fonte: O Autor (2018).

5.4 Condições de Anormalidade

Para esta etapa, visando a validação da indicação em cores diferentes de cada um dos parâmetros analisados segundo suas leituras de acordo com o padrão ou não, foram feitos os seguintes testes:

- **TA1:** Anormalidade no Tanque de Arla32
- **Objetivo:** Validar a detecção de anormalidades na leitura do sensor de volume do tanque do aditivo ARLA32.
- **Resultado:** Satisfatório
- **TA2:** Anormalidade no Nível elevado de emissões
- **Objetivo:** Validar a detecção de anormalidades na leitura do sensor de emissões do veiculo à um nível alto.
- **Resultado:** Satisfatório
- **TA3:** Anormalidade no Nível alarmante de emissões
- **Objetivo:** Validar a detecção de anormalidades na leitura do sensor de emissões do veiculo à um nível alarmante.
- **Resultado:** Satisfatório
- **TA4:** Anormalidade na falha de comunicação
- **Objetivo:** Validar a detecção de falhas de comunicação entre os sensores envolvidos no processo de pós-tratamento do veiculo.
- **Resultado:** Satisfatório
- **TA5:** Anormalidade em falhas elétricas
- **Objetivo:** Validar a detecção de falhas elétricas do veiculo no processo de pós-tratamento.
- **Resultado:** Satisfatório
- **TA6:** Anormalidade na leitura de valores inconsistentes
- **Objetivo:** Validar a detecção de valores inconsistentes nos sensores envolvidos no processo de pós-tratamento do veiculo.

- **Resultado:** Satisfatório

Antes de iniciar os 6 testes envolvendo cada um dos parâmetros a serem analisados, primeiro foi necessário a execução de uma tabela verdade para todos os casos, afim de estabelecer quais os valores e indicações que estariam dentro ou não dos padrões e quando essas leituras indicariam para o usuário algo fora do comum acontecendo. Também é importante para indicar o funcionamento ou não do *software* em relação à detecção das anormalidades. Os resultados e estabelecimento dos padrões podem ser vistos na Figura 5.8, e serão explicados na sequencia.

Figura 5.8: Tabela Verdade, Testes TA1 - TA6

Parâmetro	Comentário	Valor Padrão	Valor Irregular	Resultado do Teste
Tanque de ARLA32 Vazio	Nível de líquido no tanque do Aditivo ARLA32	>= 11%	<11%	Satisfatório
Nível Elevado de Emissões	Emissões acima do nível permitido pela legislação	0<=X<550 [ppm]	X>=550 & X<1500	Satisfatório
Nível Alarmante de Emissões	Emissões acima do nível permitido pela legislação e pelo IBAMA	0<=X<550 [ppm]	X>=1500	Satisfatório
Falha de Comunicação	Indicações de falha na comunicação dos sensores envolvidos no processo de pós-tratamento	In Range	Error / Not Available	Satisfatório
Falha Elétrica	Indicações de falha elétrica dos sensores envolvidos no processo de pós-tratamento	In Range	Error / Not Available	Satisfatório
Valor Inconsistente	Indicações de valores inconsistentes nos sensores envolvidos no processo de pós-tratamento	In Range	Error / Not Available	Satisfatório

Fonte: O Autor (2018).

Para todos os testes (TA1 TA6) foi utilizado o laboratório da empresa, onde todos os equipamentos necessários se faziam presentes, e o procedimento foi o seguinte: primeiro conetou-se o *dongle* à uma fonte de 5V e a ECU a uma fonte de 24 V para energizá-los, em seguida conectou-se a ECU a um trennadapter para que fosse possível ter acesso aos seus terminais, principalmente à alimentação e barramento CAN.

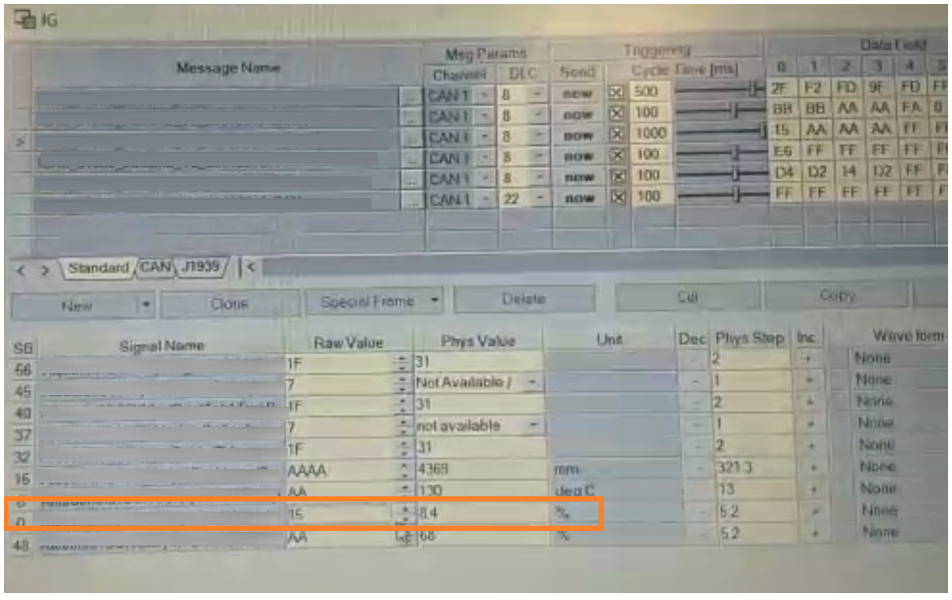
Em seguida conectou-se o barramento CAN ao dispositivo CANcase e ao *dongle* para simular a conexão direta ao veiculo e para que o dispositivo pudesse adquirir as informações passando no barramento daquela ECU. O dispositivo CANcase é então conectado ao computador, sendo o responsável por adquirir os dados da ECU e transmiti-los para um *software* de análise e construção de mensagens do barramento CAN. Neste *software* chamado CANalyzer foi inserida uma biblioteca desenvolvida pelo autor de interpretação de mensagens CAN utilizadas para propagar o protocolo SAE

J1939.

Portanto, foi possível através da interface do CANalyzer verificar quais informações estavam passando pelo barramento da ECU do caminhão e analisar os valores respectivos das mensagens relevantes que são lidas pelo aplicativo desenvolvido. Como todas as que são levadas em consideração na hora de analisar se os parâmetros do aplicativo estão ou não de acordo com o padrão estavam com suas leituras corretas, visto que o *software* da ECU utilizada é aplicado em caminhões que rodam como terceiros pela empresa, obviamente não estariam alterados. O melhor a se fazer foi executar os testes laboratorialmente, onde, através do CANalyzer, se torna possível a manipulação das mensagens do barramento.

Para o teste (TA1), inibiu-se a mensagem original relativa ao sensor de nível do aditivo do caminhão, e com o uso da biblioteca criada pelo autor criou-se uma mensagem idêntica com a possibilidade de alteração dos parâmetros dela. Então, dentro da mensagem, nos bits que representam o valor lido do volume de aditivo, alterou-se sua leitura para como se o tanque estivesse com menos de 11% do seu volume, o que caracteriza, de acordo com a legislação e a Figura 5.8, um caso irregular. Essa alteração pode ser vista na Figura 5.9.

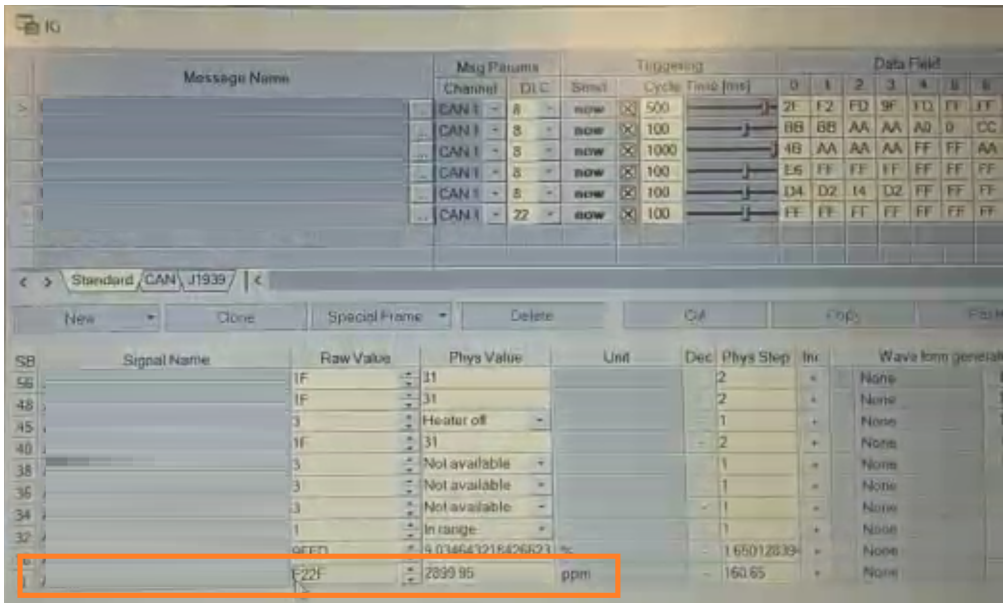
Figura 5.9: Alteração Nível do ARLA32



Fonte: O Autor (2018).

Para os testes (TA2 e TA3), inibiu-se a mensagem original relativa ao sensor de emissões do sistema de pós-tratamento, e com o uso da biblioteca criada pelo autor criou-se uma mensagem idêntica com a possibilidade de alteração dos parâmetros dela. Então, dentro da mensagem, nos bits que representam o valor lido da quantidade de emissões tóxicas em partes por milhão, alterou-se sua leitura para como se o os gases de escape do veiculo estivessem emitindo toxinas à um nível praticamente categorizado como crime ambiental. O valor escolhido foi alto justamente para poder em um exemplo só contemplar os dois indicadores de emissões presentes no aplicativo. E esse nível alarmante caracteriza, de acordo com a Figura 5.8, um caso irregular. Essa alteração pode ser vista na Figura 5.10.

Figura 5.10: Alteração Nível de emissões

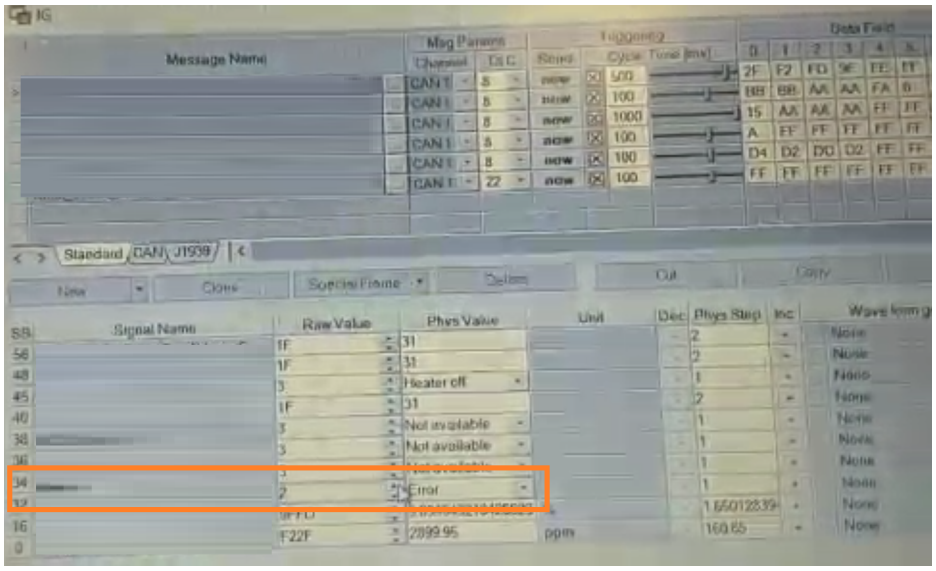


Fonte: O Autor (2018).

Para o teste (TA4 e TA5), inibiu-se a mensagem original relativa ao estado da comunicação e continuidade elétrica dos sensores envolvidos no sistema de pós-tratamento, e com o uso da biblioteca criada pelo autor criou-se uma mensagem idêntica com a possibilidade de alteração dos parâmetros dela. Então, dentro da mensagem, nos bits que representam o valor indicativo da comunicação e continuidade elétrica do sistema de sensores, alterou-se sua leitura para como se tivesse acontecido alguma interrupção externa no sistema, por exemplo para inserir um emulador, ou

alguma falha na comunicação de todos os sensores. O valor neste caso indica erro nos processos. E esse nível lógico caracteriza, de acordo com a Figura 5.8, um caso irregular. Essa alteração pode ser vista na Figura 5.11.

Figura 5.11: Alteração Falha de Comunicação e Elétrica



Fonte: O Autor (2018).

Para o teste (TA6), inibiu-se a mensagem original relativa aos valores lidos nos sensores do sistema de pós-tratamento, e com o uso da biblioteca criada pelo autor criou-se uma mensagem idêntica com a possibilidade de alteração dos parâmetros dela. Então, dentro da mensagem, alterou-se sua leitura para como se tivesse ocorrido alguma leitura com valores negativos ou inconsistentes perante à resolução do sensor. Para o teste final, alterou-se os valores do tanque de aditivo ARLA32 para patamares inferiores a 11%, alterou-se o nível de emissões em ppm para acima de 1500 e o parâmetro indicador de falha de comunicação para indicar erro.

Após essas alterações, o uso do aplicativo seguiu os seguintes procedimentos: ligou-se o aplicativo em sua tela principal, conectou-se ao *dongle*, na página de informações gerais pressionou-se o botão 'Ok', na tela de seleção de parâmetros, todos eles foram selecionados para serem analisados, menos o que indicaria o 'Valor Inconsistente'. Foi feita essa escolha para demonstrar na próxima tela que além de ler corretamente e exibir os parâmetros irregulares, não exibiria nada em relação ao

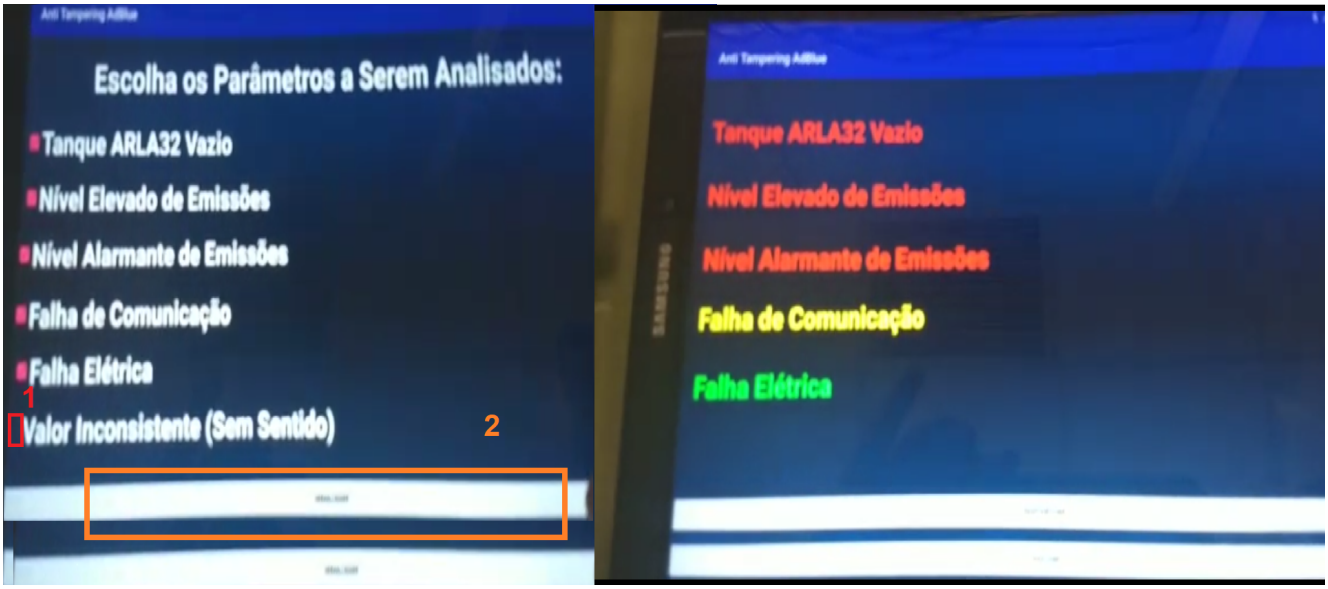
parâmetro de 'Valor Inconsistente', visto que não foi selecionado.

Pressionando o botão 'Analisar' passou-se para a página relativa à exibição dos parâmetros, em verde (caso dentro dos padrões), vermelho (fora dos padrões e grande indicador de burla) e amarelo (fora dos padrões e provável indicador de burla). Portanto, para as alterações feitas conforme citado acima, os parâmetros deveriam indicar o seguinte:

- Tanque de ARLA32 Vazio: **Irregular**
- Nível Elevado de Emissões: **Irregular**
- Nível Alarmente de Emissões: **Irregular**
- Falha de Comunicação: **Irregular**
- Falha Elétrica: **Regular**
- Valor Inconsistente: **Não Exibir**

Ao executar todos os procedimentos e comprovar, segundo a Figura 5.12, que todos os indicadores estão de acordo com o esperado, pode-se concluir que o teste foi satisfatório.

Figura 5.12: Teste Anormalidade de Parâmetros



Fonte: O Autor (2018).

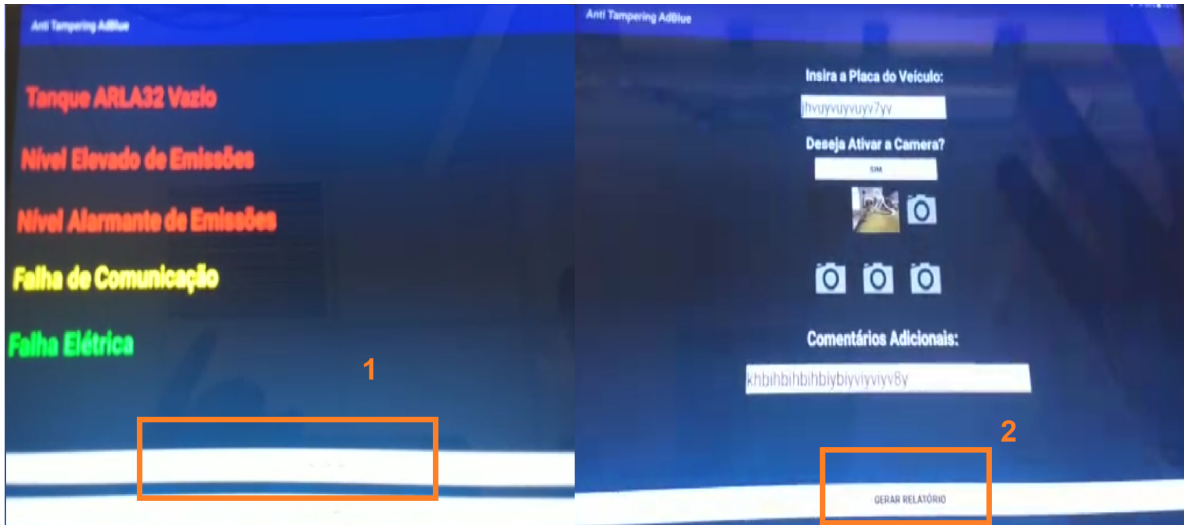
5.5 Relatório

Para esta etapa, visando a validação da aquisição dos parâmetros, informações e geração do relatório, foi feito o seguinte teste:

- **TR1:** Geração de Relatório.
- **Objetivo:** Validar a aquisição das informações, funcionamento do álbum de fotos e geração do relatório.
- **Resultado:** Regular.

Para o teste (TR1) foi utilizado o laboratório da empresa, onde todos os equipamentos necessários se faziam presentes, e o procedimento foi o seguinte: com todo o sistema funcionando, após chegar na tela de visualização e parecer da leitura dos parâmetros, pressionou-se o botão 'Reportar'. Na tela posterior buscou-se verificar se os dados inseridos nos campos de 'placa' e 'comentários' permaneciam intactos ao mudar de tela para a de edição das imagens. Também buscou-se validar o funcionamento do álbum de fotos apertando o botão 'sim' e tirando fotos com a câmera para analisar se os espaços indicados seriam preenchidos da maneira correta. Este procedimento pode ser visto na Figura 5.2.

Figura 5.13: Teste Relatório TR1



Fonte: O Autor (2018).

Após confirmação de todas as etapas do teste TR1 com funcionamento correto, o objetivo foi definir quais informações e de qual maneira ficariam dispostas no relatório em PDF gerado. Portanto, optou-se por armazenar as informações relevantes adquiridas ao decorrer de todo o uso do *software* e exibi-las da seguinte forma: abaixo do titulo 'Informações Gerais', inserir data, hora, modelo do veiculo selecionado, VIN, número de série, protocolo utilizado, placa, tempo da MIL acesa e os comentários. Seguinte à este parecer e abaixo do título de 'Inconsistências' a seguir, foi decidido por inserir quais parâmetros detectados por aquela utilização estavam fora do normal. E por fim, inserir as imagens tiradas pelo celular. O modelo de um relatório gerado pode ser visto na Figura 5.14.

Figura 5.14: Exemplo Relatório

INFORMAÇÕES GERAIS

Data: 0
Hora: 11:45
Veículo: SCANIA
VIN: 1HGBH41JXMN109186
Número de Série: 10937283098
Protocolo: SAE J1939
Placa: ATX-9999
Tempo da MIL Acesa: 100 min
Comentários Adicionais: Veiculo com tempo anormal de MIL acesa

INCONSISTÊNCIAS

Tanque ARLA32 Vazio
Nível Elevado de Emissões
Falha Elétrica

FOTOS

Fonte: O Autor (2018).

CAPÍTULO 6

CONCLUSÃO

Com o crescimento cada vez maior dos casos de burla no sistema dos caminhões que cuidam do tratamento dos gases de escape decorrentes da combustão para que sejam expelidos da maneira mais “limpa” possível, a situação vem se tornando cada vez mais alarmante, visto que os meios de fiscalização atuais de emissões são precários e pouco confiáveis. E graças a isso surge a necessidade por métodos e dispositivos mais confiáveis, que forneçam dados transparentes que possam auxiliar os policiais rodoviários e os agentes do IBAMA nestas análises, para que a legislação seja cumprida e os níveis de emissões reduzidos.

Alguns exemplos que valem ser citados para demonstrar a gravidade do caso são vistos em algumas notícias distribuídas pela internet, como por exemplo ”Utilização de ARLA 32 fica 50% abaixo do esperado para a frota do Brasil” (CAMINHONEIRO, 2017), ou também ”Caminhoneiros burlam a lei e rodam sem aditivo que reduz poluição” (GLOBO, 2016).

Tendo como base os resultados dos testes e todo o desenvolvimento feito até o momento, pode-se concluir ao observar os objetivos específicos propostos que a porcentagem de conclusão de cada um deles é a que pode ser vista na Figura 6.1.

Figura 6.1: Conclusão dos Objetivos Específicos

Objetivo Específico	Porcentagem Concluída
Estudo Normas	100%
Requisitos | Proposta	100%
Projetar Sistema	100%
Interface | Comunicação	100%
Coleta | Interpretação de Dados	100%
Criação e estrutura do relatório	100%*

Fonte: O Autor (2018).

O objetivo específico de realizar estudos sobre as normas e legislações que regulam os sistemas automotivos e de emissões de poluentes, foi concluído pois diversas normas, protocolos e legislações tiveram de ser estudadas e aplicadas no projeto. Um dos principais protocolos dos sistemas automotivos e também o principal protocolo necessário para o desenvolvimento do projeto, o protocolo CAN, teve que ser profundamente estudado, os protocolos J1939 e OBD-II de modo análogo tiveram que ser estudados. As legislações de emissões também foram profundamente estudadas, no caso da Europa, EURO V, e no Brasil a Proconve P7.

A etapa de analise de requisitos e elaboração da proposta também foi completa em seu todo, visto que no decorrer do desenvolvimento do projeto, foram feitos alguns *user experiences* e reuniões com a PRF para estudar quais requisitos eram necessários para o projeto, bem como a elaboração da proposta apresentada pelo trabalho. Também foram realizadas reuniões de *feedback* para teste do modelo *dummy* do *software* e melhoramento da proposta segundo opinião dos próprios usuários.

Como dito no parágrafo anterior, graças a todas as reuniões, *user experiences* e *feedbacks* recebidos, foi possível também completar o objetivo especifico de projetar todo o sistema. Além do contato com o cliente, vale salientar que neste processo foram feitas diversas pesquisas de viabilidade dos componentes de *hardware* (*dongle* e cabos), e levado em consideração a aplicação dos sistemas de *software* (android e programação JAVA).

A implementação da interface e da comunicação foi totalmente implementada conforme o planejado. A ideia era a de fazer um aplicativo que em tempo de execução fosse possível analisar todos os parâmetros acordados com o cliente que seriam relevantes para uma possível detecção de burla no sistema de pós-tratamento e o resultado foi satisfatório, visto que após melhoria do código e uso de *threads*, a aplicação é capaz de adquirir e analisar todas as informações em um intervalo imperceptível aos olhos do usuário. A interface também foi implementada de maneira satisfatória, pois, com os *feedbacks* dos usuários pode-se melhorar seu design e termos utilizados ao longo da aplicação para tornar o processo mais intuitivo.

O objetivo específico relacionado à coleta e interpretação dos dados foi concluído de maneira satisfatória, pois, após os estudos sobre a estruturação do barramento CAN, comunicação e aplicação da funcionalidade *Bluetooth*, forma de transmissão dos dados pelo *dongle*, bem como dos protocolos J1939 e OBD-II tornou-se possível a coleta correta dos dados e também sua interpretação, sendo possível em futuras atualizações adicionar ou retirar mensagens a serem analisadas. Ao conhecer mais e validar o formato de recebimento das mensagens foi possível também realizar o tratamento delas, analisando cada parte de sua estrutura e extraindo as informações relevantes para usá-las da maneira mais correta na hora do funcionamento e análise dos parâmetros no aplicativo. Foi também possível com essa extração de dados a realização da validação dos parâmetros lidos e dos testes de funcionamento do aplicativo já com a parte de leitura real de informações realizada. Os resultados foram satisfatórios e a validação dos métodos e *hardware* envolvidos no processo também atendeu às expectativas.

Para afirmar com mais propriedade sobre o êxito na etapa de coleta e interpretação de dados, os procedimentos de testes foram fundamentais. Os testes, tanto em laboratório quanto em veiculo estão marcados como completos com uma ressalva, dado que o projeto ainda se encontra em fase de melhoria, aprimoramento e seleção dos dados. Em veiculo, os testes foram relativamente satisfatórios, pois, só foi possível e permitido realizar os testes em caminhões de descarga que parassem dentro da empresa, então obviamente nenhum deles teria algum mecanismo de burla em seu sistema. Porém, com o uso das ferramentas corretas, foi possível pegar alguns minutos de *log* do próprio *software* destes caminhões e em laboratório realizar a inibição e alteração de algumas mensagens para simular um caminhão rodando em condições de burla e verificar se a aplicação detectaria e informaria ao usuário de maneira correta. Portanto, os testes em laboratório também foram satisfatórios pois em todos os cenários de simulação responderam corretamente.

Portanto, a ressalva feita no tópico de testes é que apesar de em bancada a aplicação ter sido testada e seu funcionamento ter sido comprovado com sucesso, os testes em caminhões não foram viáveis por falta de oportunidade de pegar uma burla

real, visto que não havia caminhão com essas características disponíveis.

6.1 Trabalhos Futuros

Foram identificadas melhorias futuras no projeto:

- I. Desenvolvimento de um painel *web based* para consulta dos resultados dos diagnósticos realizados;
- II. Portabilidade para sistemas IOS;

Para seguir com os processos de melhoria do projeto, a primeira parte é a de pesquisar formas para migrar a aplicação de *software* também para plataformas baseadas em sistemas IOS, assim o projeto terá a possibilidade de abranger mais usuários e de disponibilizar a flexibilidade para que utilizem o projeto com a plataforma com que possuírem mais afinidade.

Outro ponto de melhoria seria o desenvolvimento de uma aplicação *web*, ou seja, um portal na internet que pudesse ser a 'central' de informações entre os usuários do sistema. Um local no qual seria possível a visualização de como está o desempenho do *software* para os usuários, centralizar as informações dos relatórios gerados para eventuais consultas e disponibilizar um meio de comunicação entre os usuários de diversas partes do Brasil para que possam ajudar uns aos outros.

REFERÊNCIAS

ALMEIDA, I. *Automotive World*. 2013. <https://irvingalmeida.wordpress.com/2013/10/06/sistema-scr-dos-motores-a-diesel/>. Acesso em 25 Ago. 2017.

ANFAVEA. *Diesel e Emissões*. 2012. <http://www.anfavea.com.br/docs/cartilha-proconveP7.pdf>. Acesso em 01 Set. 2017.

BARRETO, V. *What is OBD II? History of On-Board Diagnostics*. 2017. <https://www.geotab.com/blog/obd-ii/>. Acesso em 01 Set. 2018.

BEAL, V. *Application (Application Software)*. 2016. <http://www.webopedia.com/TERM/A/application.html>. Acesso em 9 Set. 2017.

CAMINHONEIRO, B. do. *Utilização de ARLA 32 fica 50% abaixo do esperado para a frota do Brasil*. 2017. <http://blogdocaminhoneiro.com/2017/08/utilizacao-de-arla-32-fica-50-abaixo-do-esperado-para-a-frota-do-brasil/>. Acesso em 9 Set. 2017.

DELPHI, D. W. *Engineering electrical/electronic architecture for today's high-tech vehicles*. 2007. <https://www.electronicdesign.com/automotive/engineering-electricalelectronic-architecture-todays-high-tech-vehicles>. Acesso em 30 Ago. 2018.

ENGINEER, T. *Vehicle Electronics*. 2014. <http://www.transportengineer.org.uk/article-images/61707/Vehicle$_{e}lectronics.pdf>.Acessoem01Set.2018.$

FENDELMAN, A. *Bluetooth Technology Overview*. 2017. <https://www.lifewire.com/definition-of-bluetooth-technology-578667>. Acesso em 10 Set. 2017.

GLOBO. *Caminhoneiros burlam lei e rodam sem aditivo que reduz poluição*. 2016. <http://g1.globo.com/fantastico/videos/t/edicoes/v/caminhoneiros-burlam-lei-e-rodam-sem-aditivo-que-reduz-poluicao/3831162//>. Acesso em 10 Set. 2017.

GUIDE, F. J. *History of Java programming language*. 2016. <http://www.freejavaguide.com/history.html>. Acesso em 10 Set. 2017.

INSTRUMENTS, T. *Introduction to the Controller Area Network (CAN)*. 2016. <http://www.ti.com/lit/an/sloa101b/sloa101b.pdf>. Acesso em 13 Mar. 2018.

IVECO. *O Que é Proconve P7*. 2011. <http://www.blogiveco.com.br/o-que-e-o-proconve-p7/>. Acesso em 05 Set. 2017.

MANAVELLA, H. *Automotive World*. 2015. <http://www.oficinabrasil.com.br/noticia/tecnicas/cenario-do-pos-tratamento-em-motores-ciclo-diesel-entenda-o-processo-de-emissoes>. Acesso em 08 Set. 2017.

MARY, R. *Bluetooth Technology*. 2017. <https://www.engineersgarage.com/articles/bluetooth-technology>. Acesso em 10 Set. 2017.

MULLIS, A. *Android Studio tutorial for beginners*. 2017. <http://www.androidauthority.com/android-studio-tutorial-beginners-637572>. Acesso em 6 Set. 2017.

NEWSLETTER, C. *20 Years CANalyzer*. 2012. <https://can-newsletter.org/tools/bus-analyzers/nr$_{b}us_{v}ector_{2}0years_{1}20417>.Acessoem13Mai.2018.$

PIANEGONDA, N. *Consumo de Arla 32 está 45% abaixo do estimado e preocupa mercado*. 2017. <http://www.cnt.org.br/Imprensa/noticia/consumo-de-arla-32-esta-45-abaixo-do-estimado-e-preocupa-mercado>. Acesso em 25 Ago. 2018.

QUEIROLO, G. *Empresas têm 60% do transporte rodoviário*. 2018. <https://www1.folha.uol.com.br/mercado/2018/05/empresas-tem-60-do-transporte-rodoviario.shtml>. Acesso em 25 Ago. 2018.

SAE. *OBD-II Parameters*. 2017. <https://www.sae.org/>. Acesso em 10 Set. 2018.

SILVA, S. D. *Internet das coisas: 34 bilhões de dispositivos em 2020*. 2016. <http://www.meioemensagem.com.br/home/marketing/2016/06/01/internet-das-coisas-34-bi-de-dispositivos-em-2020.html>. Acesso em 9 Ago. 2017.

SOFTWARE, S. *VNA-232 Simma Software*. 2016. <http://www.simmasoftware.com/j1939-to-rs232.pdf>. Acesso em 13 Mar. 2018.

SOLUTIONS, O. *What is OBD?* 2016. <https://www.obdsol.com/knowledgebase/on-board-diagnostics/what-is-obd/>. Acesso em 25 Ago. 2018.

SPONAS, J. G. *Java Object Oriented Programming concepts*. 2016. <http://blog.nordicsemi.com/getconnected/things-you-should-know-about-bluetooth-range>. Acesso em 20 Set. 2017.

TECHOPEDIA. *Application (Application Software)*. 2017. <https://www.techopedia.com/definition/4224/application-software>. Acesso em 9 Set. 2017.

TESTING, E. *ECU Explained*. 2016. <http://www.ecutesting.com/ecu.html>. Acesso em 01 Set. 2018.

VECTOR. *Introduction to J1939*. 2010. <https://vector.com/portal/medien/cmc/application$_notes/AN-ION-1-3100_Introduction_to_J1939.pdf>. Acesso em 13 Mar. 2018.

VECTOR. *Variants of CANalyzer*. 2010. <https://vector.com/vi$_canalyzer_en.html!vi_canalyzer_variants_iframe_en.html>. Acesso em 13 Mai. 2018.

VECTOR. *Variants of CANalyzer*. 2013. $<$https://vector.com/vi$_c analyzer_e n.html!vi_c analyzer_g uideme_i frame_e n.html>. Acessoem 13Mai.2018.

VINíCIUS, T. *Java: história e principais conceitos*. 2016. $<$http://www.devmedia.com.br/java-historia-e-principais-conceitos/25178$>$. Acesso em 12 Set. 2017.

W3RESOURCE. *Java Object Oriented Programming Concepts*. 2017. $<$https://www.w3resource.com/java-tutorial/java-object-oriented-programming.php$>$. Acesso em 29 Set. 2017.

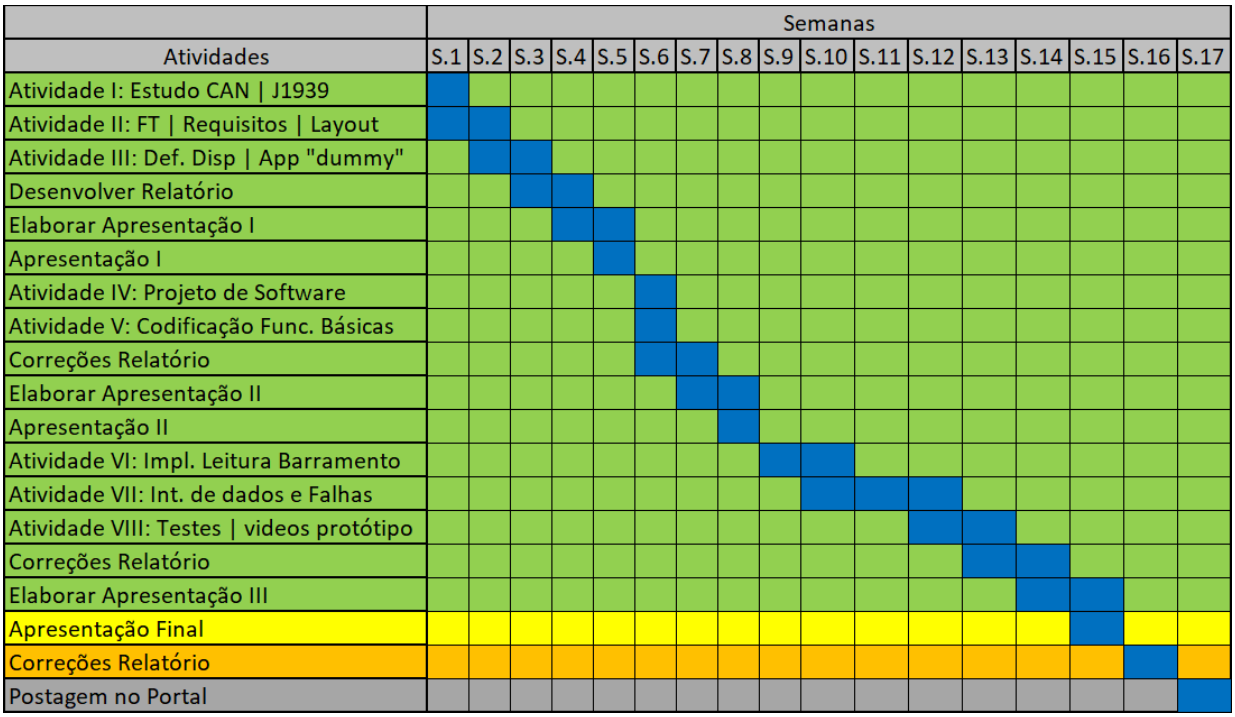
WENBO, Y.; QUANYU, W.; ZHENWEI, G. Smart home implementation based on internet and wifi technology. In: IEEE. *Control Conference (CCC), 2015 34th Chinese*. [S.l.], 2015. p. 9072–9077.

WIKIPEDIA. *Dongle*. 2015. $<$https://en.wikipedia.org/wiki/Dongle$>$. Acesso em 13 Mar. 2018.

APÊNDICE A

CRONOGRAMA

Figura A.1: Cronograma



Fonte: O Autor (2018).

APÊNDICE B

IDEALIZAÇÃO DO LAYOUT

Figura B.1: Tela Inicial e do Call Center

Fonte: O Autor (2018).

Figura B.2: Tela de Atualização e de Seleção de Parâmetros

Fonte: O Autor (2018).

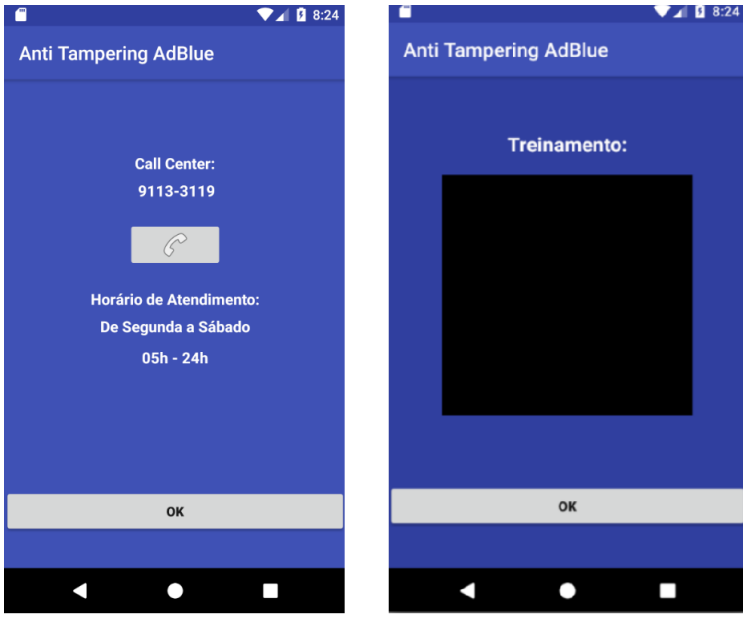
Figura B.3: Tela de Análise e de Relatório

Fonte: O Autor (2018).

APÊNDICE C

LAYOUTS FINAIS

Figura C.1: Call Center e Treinamento

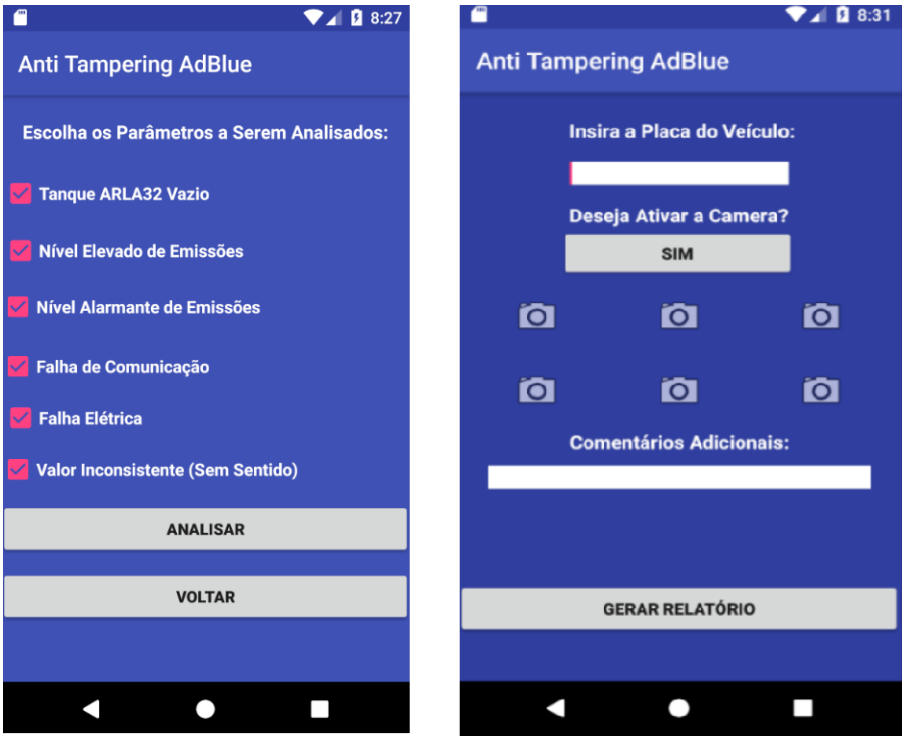


Fonte: O Autor (2018).

Figura C.2: Tela Inicial e Informações Gerais

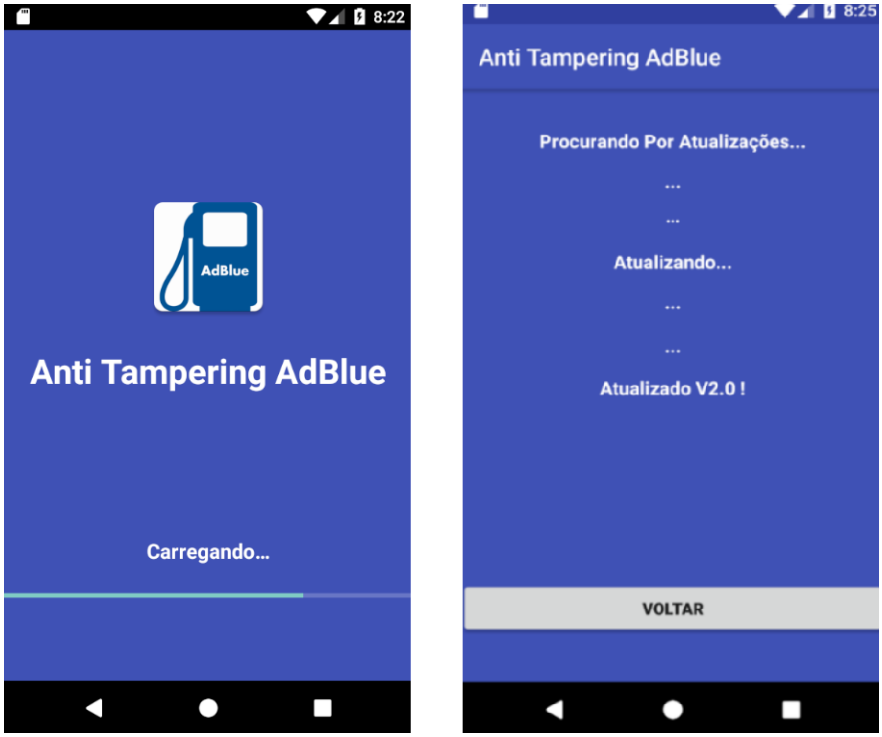
Fonte: O Autor (2018).

Figura C.3: Escolha de Prâmetros e Relatório



Fonte: O Autor (2018).

Figura C.4: Carregamento e Atualização



Fonte: O Autor (2018).

Figura C.5: Exibição de Parâmetros

Fonte: O Autor (2018).

APÊNDICE D

MANIFESTO

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.antiburla">

    <uses-permission android:name="android.permission.CALL_PHONE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.BLUETOOTH"/>
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-permission android:name="android.permission.CAMERA"/>
    <uses-feature android:name="android.hardware.camera" android:required="true"/>

    <application
        android:allowBackup="true"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/Theme.AppCompat.NoActionBar"
        >

        <activity android:name=".SplashActivity"
            android:screenOrientation="portrait">
            <intent-filter>
```

```
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>

    <activity android:name=".MainActivity"
        android:screenOrientation="portrait"
        android:theme="@style/AppTheme"/>
    <activity android:name=".CallActivity"
        android:screenOrientation="portrait"
        android:theme="@style/AppTheme"/>
    <activity android:name=".VeriffActivity"
        android:screenOrientation="portrait"
        android:theme="@style/AppTheme"/>
    <activity android:name=".UpdateActivity"
        android:screenOrientation="portrait"
        android:theme="@style/AppTheme"/>
    <activity android:name=".ChooseActivity"
        android:screenOrientation="portrait"
        android:theme="@style/AppTheme"/>
    <activity android:name=".InformationActivity"
        android:screenOrientation="portrait"
        android:theme="@style/AppTheme"/>
    <activity android:name=".ShowActivity"
        android:screenOrientation="portrait"
        android:theme="@style/AppTheme"/>
    <activity android:name=".TrainingActivity"
        android:screenOrientation="portrait"
```

```
            android:theme="@style/AppTheme"/>
        <activity android:name=".ImageActivity"
            android:screenOrientation="portrait"
            android:theme="@style/AppTheme"/>
        <activity android:name=".DeviceListActivity"
            android:label="@string/txt_selecdisp"
            android:theme="@android:style/Theme.Dialog"/>

        <provider
            android:name="android.support.v4.content.FileProvider"
            android:authorities="${applicationId}.provider"
            android:exported="false"
            android:grantUriPermissions="true">
            <meta-data
                android:name="android.support.FILE_PROVIDER_PATHS"
                android:resource="@xml/provider_paths"/>
        </provider>

    </application>

</manifest>
```

APÊNDICE E

CLASSE - CALLACTIVITY

```
package com.antitamperingadblue;


        import android.Manifest;
        import android.content.Intent;
        import android.content.pm.PackageManager;
        import android.net.Uri;
        import android.os.Bundle;
        import android.support.v4.app.ActivityCompat;
        import android.support.v4.content.ContextCompat;
        import android.support.v7.app.AppCompatActivity;
        import android.view.View;
        import android.view.View.OnClickListener;
        import android.widget.Button;
        import android.widget.ImageButton;



public class CallActivity extends AppCompatActivity {
    //-------------------------Declarao de botes----------------------//
    Button btnhome;
    ImageButton btncall;
    private static final int REQUEST_CALL = 1;
    Intent callIntent;


    //-------------------------Inicializao do Layout----------------//
```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.call_layout);
    if (ContextCompat.checkSelfPermission(CallActivity.this,
    Manifest.permission.CALL_PHONE) != PackageManager.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions(CallActivity.this, new String[]{
                Manifest.permission.CALL_PHONE}, REQUEST_CALL);
    }
    callProcess();
    homeProcess();
}

//---------------Ao Clicar No Boto Liga P/ o Nmero---------------//
public void callProcess() {
    btncall = (ImageButton) findViewById(R.id.btn_call);
    btncall.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            callIntent = new Intent(Intent.ACTION_CALL);
            callIntent.setData(Uri.parse("tel:+55(41)9113-3119"));
            startActivity(callIntent);
        }
    });
}


//---------Ao Clicar No Boto Mudar Para Tela Inicial-------------//
public void homeProcess() {
```

```
        btnhome = (Button) findViewById(R.id.btn_ok3);
        btnhome.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent home = new Intent(CallActivity.this, MainActivity.class);
                startActivity(home);
                finish();
            }
        });
    }
}
```

APÊNDICE F

CLASSE - CHOOSEACTIVITY

```
package com.antitamperingadblue;


        import android.content.Intent;
        import android.os.Bundle;
        import android.support.v7.app.AppCompatActivity;
        import android.view.View;
        import android.widget.Button;
        import android.widget.CheckBox;



public class ChooseActivity extends AppCompatActivity{
    //--------------Declarao de Botes e Variveis-------------------------//
    Button btnnext;
    Button btnback;

    int flag_interrupt = 1, flag_tanque= 1, flag_nox3 = 1, flag_nox7 = 1,
        flag_falha = 1, flag_plaus = 1;

    //-------------------Inicializao do Layout----------------------------//
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.choose_param);
```

```
        Intent infoIntent = getIntent();
        final String NOXS = infoIntent.getStringExtra("NOXS");
        final String TANKS = infoIntent.getStringExtra("TANKS");
        final String COMS = infoIntent.getStringExtra("COMS");
        final String TANKF = infoIntent.getStringExtra("TANKF");
        final String NOX3F = infoIntent.getStringExtra("NOX3F");
        final String NOX7F = infoIntent.getStringExtra("NOX7F");
        final String COMF = infoIntent.getStringExtra("COMF");
        final String PLAUF = infoIntent.getStringExtra("PLAUF");
        final String intElet = infoIntent.getStringExtra("intElet");
        final String protStamp = infoIntent.getStringExtra("protStamp");
        final String milStamp = infoIntent.getStringExtra("milStamp");
        final String vinStamp = infoIntent.getStringExtra("vinStamp");
        final String esnStamp = infoIntent.getStringExtra("esnStamp");
        final String vehiStamp = infoIntent.getStringExtra("vehiStamp");

//-------------Ao Clicar No Boto Mudar Para Prxima Tela-----------------//

        btnnext = (Button) findViewById(R.id.btn_anal);
        btnnext.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                Intent analise = new Intent(ChooseActivity.this,
                ShowActivity.class);

/---Adiciona os Valores das Flags ao Pacote de Envio Para ShowActivity----//

                analise.putExtra("flag_interrupt", flag_interrupt);
```

```
                analise.putExtra("flag_tanque", flag_tanque);
                analise.putExtra("flag_nox3", flag_nox3);
                analise.putExtra("flag_nox7", flag_nox7);
                analise.putExtra("flag_falha", flag_falha);
                analise.putExtra("flag_plaus", flag_plaus);
                analise.putExtra("protStamp", protStamp);
                analise.putExtra("milStamp", milStamp);
                analise.putExtra("avinStamp", vinStamp);
                analise.putExtra("esnStamp", esnStamp);
                analise.putExtra("vehiStamp", vehiStamp);
                analise.putExtra("TANKS", TANKS);
                analise.putExtra("TANKF", TANKF);
                analise.putExtra("NOX3F", NOX3F);
                analise.putExtra("NOX7F", NOX7F);
                analise.putExtra("COMF", COMF);
                analise.putExtra("PLAUF", PLAUF);
                analise.putExtra("NOXS", NOXS);
                analise.putExtra("TANKS", TANKS);
                analise.putExtra("COMS", COMS);
                analise.putExtra("intElet", intElet);
                startActivity(analise);
                finish();
            }
        });
//-----------Ao Clicar No Boto Mudar Para Tela Inicial-----------------//
        btnback = (Button) findViewById(R.id.btn_volt);
        btnback.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
```

```
                Intent home = new Intent(ChooseActivity.this, MainActivity.class);
                startActivity(home);
                finish();
            }
        });
    }
//-----Verifica se o CheckBox Est Setado Retornando um Booleano--------//
    public void selectItem(View view){
        boolean checked = ((CheckBox)view).isChecked();

//------A Partir do Booleano Seta-se ou No a Flag de Cada CheckBox------//
        switch (view.getId()){

            case R.id.check_interrup:
                if(checked){flag_tanque = 1;} else {flag_tanque = 0;} break;
            case R.id.check_tanque:
                if(checked){flag_nox3 = 1;}   else {flag_nox3 = 0;}   break;
            case R.id.check_nox3_5:
                if(checked){flag_nox7 = 1;}   else {flag_nox7= 0;}    break;
            case R.id.check_nox7_0:
                if(checked){flag_falha = 1;}  else {flag_falha = 0;}  break;
            case R.id.check_falha:
                if(checked){flag_interrupt = 1;} else {flag_interrupt = 0;} break;
            case R.id.check_plau:
                if(checked){flag_plaus = 1;}  else {flag_plaus = 0;}  break;
        }
    }
}
```

APÊNDICE G

CLASSE - DEVICELISTACTIVITY

```
package com.antitamperingadblue;


        import android.app.Activity;
        import android.bluetooth.BluetoothAdapter;
        import android.bluetooth.BluetoothDevice;
        import android.content.BroadcastReceiver;
        import android.content.Context;
        import android.content.Intent;
        import android.content.IntentFilter;
        import android.os.Bundle;
        import android.view.View;
        import android.view.Window;
        import android.widget.AdapterView;
        import android.widget.ArrayAdapter;
        import android.widget.Button;
        import android.widget.ListView;
        import android.widget.TextView;
        import java.util.Set;



public class DeviceListActivity extends Activity {

    // Retornar Extras da Intent
    public static final String EXTRA_DEVICE_ADDRESS = "device_address";
```

```
// Parmetros
private BluetoothAdapter bluettoothAdapter;
private ArrayAdapter<String> mNovosDispositivos;
private ArrayAdapter<String> mDispositivosPareados;


@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // ConfigurandoTela
    requestWindowFeature(Window.FEATURE_INDETERMINATE_PROGRESS);
    setContentView(R.layout.device_list);

    // Caso o Usurio cancele a operao
    setResult(Activity.RESULT_CANCELED);

    // Inizializao do boto para procurar dispositivos
    Button btnscan = (Button) findViewById(R.id.btn_scan);
    btnscan.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            doDiscovery();
            v.setVisibility(View.GONE);
        }
    });

    // Inicializa os ArrayAdapters.
```

```
mDispositivosPareados = new ArrayAdapter<>(this, R.layout.device_view);
mNovosDispositivos = new ArrayAdapter<>(this, R.layout.device_view);


// Encontrar e Configurar o ListView para dispositivos pareados
ListView listaPareados = (ListView) findViewById(R.id.paired_devices);
listaPareados.setAdapter(mDispositivosPareados);
listaPareados.setOnItemClickListener(mDeviceClickListener);

// Encontrar e Configurar o ListView para novos dispositivos
ListView listaNovos = (ListView) findViewById(R.id.new_devices);
listaNovos.setAdapter(mNovosDispositivos);
listaNovos.setOnItemClickListener(mDeviceClickListener);

// Registrar para broadcast quando um dispositivo for descoberto
IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
this.registerReceiver(mReceiver, filter);

// Registrar para broadcast quando a descoberta acabar
filter = new IntentFilter(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
this.registerReceiver(mReceiver, filter);

// BluetoothAdapter Local
bluettoothAdapter = BluetoothAdapter.getDefaultAdapter();

// Adquirir os dispositivos j pareados
Set<BluetoothDevice> dispsPareados = bluettoothAdapter.getBondedDevices();

//Se existirem dispositivos pareados, adicion-los no ArrayAdapter
```

```
    if (dispsPareados.size() > 0){
        findViewById(R.id.title_paired_devices).setVisibility(View.VISIBLE);
        for (BluetoothDevice device : dispsPareados){
            mDispositivosPareados.add(device.getName()+ "\n"
            + device.getAddress());
        }
    }
    else{
        String semDisp = getResources().getText(R.string.txt_semdisp)
        .toString();
        mDispositivosPareados.add(semDisp);
    }
}

@Override
protected void onDestroy()
{
    super.onDestroy();

    // Garantir que o processo de descoberta seja encerrado
    if (bluettoothAdapter != null)
    {
        bluettoothAdapter.cancelDiscovery();
    }

    // Apagar os listeners de broadcast
    this.unregisterReceiver(mReceiver);
}
```

```
private void doDiscovery(){
    // Indicar o titulo de procurando
    setProgressBarIndeterminateVisibility(true);
    setTitle(R.string.txt_procdisp);

    // Liberar subtulo para novos dispositivos
    findViewById(R.id.title_new_devices).setVisibility(View.VISIBLE);

    // Se j estiver descobrindo, parar
    if (bluettoothAdapter.isDiscovering()){
        bluettoothAdapter.cancelDiscovery();
    }

    // Comear processo de descobertas
    bluettoothAdapter.startDiscovery();
}

// Processo de deteco de clique paraod ListViews
private final AdapterView.OnItemClickListener mDeviceClickListener =
            new AdapterView.OnItemClickListener()
{
    public void onItemClick(AdapterView<?> av, View v, int arg2, long arg3)
    {
        // Cancelar descobertas porque demanda muito processamento
        bluettoothAdapter.cancelDiscovery();

        // Conseguir endereos MAC
        String info = ((TextView) v).getText().toString();
        String address = info.substring(info.length() - 17);
```

```
            // Criar o pacote de resultante com o endereo MAC
            Intent intent = new Intent();
            intent.putExtra(EXTRA_DEVICE_ADDRESS, address);
            setResult(Activity.RESULT_OK, intent);
            finish();
        }
    };


// O BroadcastReciever que procura por dispositivos e muda o titulo
    private final BroadcastReceiver mReceiver = new BroadcastReceiver()
    {
        @Override
        public void onReceive(Context context, Intent intent)
        {
            String action = intent.getAction();

            // Quando um dispositivo  descoberto
            if (BluetoothDevice.ACTION_FOUND.equals(action))
            {
                BluetoothDevice device =
                intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
                // Se j estiver pareado, pula dispositivo
                if (device.getBondState() != BluetoothDevice.BOND_BONDED)
                {
                    mNovosDispositivos.add(device.getName() + "\n"
                    + device.getAddress());
```

```
                }
                // Quando a descoberta termina, muda o titulo da atividade
            }
            else if (BluetoothAdapter.ACTION_DISCOVERY_FINISHED.equals(action))
            {
                setProgressBarIndeterminateVisibility(false);
                setTitle(R.string.txt_selecdisp);
                if (mNovosDispositivos.getCount() == 0)
                {
                    String noDevices = getResources()
                    .getText(R.string.txt_semdisp).toString();
                    mNovosDispositivos.add(noDevices);
                }
            }
        }
    };
}
```

APÊNDICE H

CLASSE - IMAGEACTIVITY

```
package com.antitamperingadblue;

        import android.content.Intent;
        import android.graphics.Bitmap;
        import android.graphics.BitmapFactory;
        import android.os.Bundle;
        import android.support.v7.app.AppCompatActivity;
        import android.view.View;
        import android.widget.Button;
        import android.widget.ImageView;

        import java.util.ArrayList;



public class ImageActivity extends AppCompatActivity {
    //-------Declarao de botes-------------------------//
    Button btndel;
    Button btnback;
    ImageView imageView;
    ArrayList<String> failStamp;

    //-----------Inicializao do Layout-------------------//
    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
super.onCreate(savedInstanceState);
setContentView(R.layout.fullimage_layout);

failStamp = new ArrayList<>();

Intent imagemIntent = getIntent();

final String placatxt = imagemIntent.getStringExtra("placatxt");
final String comentxt = imagemIntent.getStringExtra("comentxt");
final String dateStamp = imagemIntent.getStringExtra("dateStamp");
final String hourStamp = imagemIntent.getStringExtra("hourStamp");
final String vehiStamp = imagemIntent.getStringExtra("vehiStamp");
final String vinStamp = imagemIntent.getStringExtra("vinStamp");
final String esnStamp = imagemIntent.getStringExtra("esnStamp");
final String protStamp = imagemIntent.getStringExtra("protStamp");
final String milStamp = imagemIntent.getStringExtra("milStamp");
failStamp = imagemIntent.getStringArrayListExtra("failStamp");

final int flg = imagemIntent.getIntExtra("flag", 0);

imageView = (ImageView) findViewById(R.id.full_image);

Bundle b = imagemIntent.getExtras();
final String[] paths = b.getStringArray("paths");

imageView.setImageBitmap(getBitmapFromPath(paths[flg]));
```

```
//------Ao Clicar No Boto Deleta a Foto e Volta Para a Tela Anterior----------//
        btndel = (Button) findViewById(R.id.btn_del);
        btndel.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                String[] pathsReturn = paths;
                pathsReturn[flg] = null;

                Intent homedel = new Intent(ImageActivity.this,
                InformationActivity.class);
                homedel.putExtra("paths", pathsReturn);
                homedel.putExtra("placatxt", placatxt);
                homedel.putExtra("comentxt", comentxt);
                homedel.putExtra("protStamp", protStamp);
                homedel.putExtra("milStamp", milStamp);
                homedel.putExtra("avinStamp", vinStamp);
                homedel.putExtra("esnStamp", esnStamp);
                homedel.putExtra("vehiStamp", vehiStamp);
                homedel.putExtra("failStamp", failStamp);
                homedel.putExtra("dateStamp", dateStamp);
                homedel.putExtra("hourStamp", hourStamp);
                startActivity(homedel);
                finish();
            }
        });

//----------Ao Clicar No Boto Volta Para a Tela Anterior----------------//
        btnback = (Button) findViewById(R.id.btn_volt6);
```

```
        btnback.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent home = new Intent(ImageActivity.this, InformationActivity.clas
                home.putExtra("paths", paths);
                home.putExtra("placatxt", placatxt);
                home.putExtra("comentxt", comentxt);
                home.putExtra("protStamp", protStamp);
                home.putExtra("milStamp", milStamp);
                home.putExtra("avinStamp", vinStamp);
                home.putExtra("esnStamp", esnStamp);
                home.putExtra("vehiStamp", vehiStamp);
                home.putExtra("failStamp", failStamp);
                home.putExtra("dateStamp", dateStamp);
                home.putExtra("hourStamp", hourStamp);
                startActivity(home);
                finish();
            }
        });
    }
    //------------------Funo Para Redimensionar Imagem--------------//
    public Bitmap getBitmapFromPath(String path) {
        BitmapFactory.Options opt = new BitmapFactory.Options();
        opt.inSampleSize = 2;

        return BitmapFactory.decodeFile(path, opt);
    }
}
```

APÊNDICE I

CLASSE - INFORMATIONACTIVITY

```
package com.antitamperingadblue;


        import android.Manifest;
        import android.content.Intent;
        import android.content.pm.PackageManager;
        import android.graphics.Bitmap;
        import android.graphics.BitmapFactory;
        import android.net.Uri;
        import android.os.Bundle;
        import android.os.Environment;
        import android.provider.MediaStore;
        import android.support.v4.app.ActivityCompat;
        import android.support.v7.app.AppCompatActivity;
        import android.view.View;
        import android.widget.Button;
        import android.widget.EditText;
        import android.widget.ImageView;
        import android.widget.Toast;


        import com.itextpdf.text.BaseColor;
        import com.itextpdf.text.Document;
        import com.itextpdf.text.DocumentException;
        import com.itextpdf.text.Font;
        import com.itextpdf.text.FontFactory;
```

```
import com.itextpdf.text.Image;
import com.itextpdf.text.PageSize;
import com.itextpdf.text.Paragraph;
import com.itextpdf.text.pdf.PdfPTable;
import com.itextpdf.text.pdf.PdfWriter;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;

public class InformationActivity extends AppCompatActivity {

  //------------Declarao de Utilitrios-------------------//
    Button btnnext, btncam;
    String mLocalImagem = "";
    static final int CAM_REQUEST = 1;
    ImageView[] imageViews = new ImageView[6];
    String[] paths;
    String placatxt, comentxt;
    ArrayList<String> failStamp;
    String dateStamp, hourStamp, vehiStamp, vinStamp,
           esnStamp, protStamp, milStamp;

    //-----------Inicializao do Layout--------------------//
    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.information_layout);

        failStamp = new ArrayList<>();

        Intent intentInfos = getIntent();

        dateStamp = new SimpleDateFormat("ddMMyyyy").format(new Date());
        hourStamp = new SimpleDateFormat("HHmmss").format(new Date());
        vehiStamp = intentInfos.getStringExtra("vehiStamp");
        vinStamp = intentInfos.getStringExtra("vinStamp");
        esnStamp = intentInfos.getStringExtra("esnStamp");
        protStamp = intentInfos.getStringExtra("protStamp");
        milStamp = intentInfos.getStringExtra("milStamp");
        failStamp = intentInfos.getStringArrayListExtra("failStamp");


//---------------Criao do PDF----------------------------------//

        btnnext = (Button) findViewById(R.id.btn_relat);
        final String finalDateStamp = dateStamp;
        final String finalHourStamp = hourStamp;
        final String finalVehiStamp = vehiStamp;
        final String finalVinStamp = vinStamp;
        final String finalEsnStamp = esnStamp;
        final String finalProtStamp = protStamp;
        final String finalMilStamp = milStamp;
        btnnext.setOnClickListener(new View.OnClickListener() {
            @Override
```

```
public void onClick(View v) {
    Intent report = new Intent(InformationActivity.this,
     MainActivity.class);

    Document pdfReport = new Document();

    String timeStamp =
    new SimpleDateFormat("ddMMyyyy_HHmmss").format(new Date());
    String nameReport = "AdBlueReport_" + timeStamp + "_";

    try{

        String outPath =
        Environment.getExternalStorageDirectory()
        + nameReport + ".pdf";
        PdfWriter.getInstance(pdfReport,
        new FileOutputStream(outPath));

        pdfReport.open();

        pdfReport.setPageSize(PageSize.A4);
        Font fontbold1 = FontFactory.getFont("Times-Roman",
        16, Font.BOLD);
        fontbold1.setColor(BaseColor.BLACK);

        pdfReport.add(addTitle1());
        Paragraph esp = new Paragraph();
        esp.add(" ");
```

```
Paragraph dados = new Paragraph();
dados.add(new Paragraph("Data: " + finalDateStamp,
fontbold1));
dados.add(new Paragraph("Hora: " + finalHourStamp,
fontbold1));
dados.add(new Paragraph("Veculo: " + finalVehiStamp,
fontbold1));
dados.add(new Paragraph("VIN: " + finalVinStamp,
fontbold1));
dados.add(new Paragraph("Nmero de Srie: "
+ finalEsnStamp, fontbold1));
dados.add(new Paragraph("Protocolo: " + finalProtStamp,
fontbold1));
dados.add(new Paragraph("Placa: " + placatxt,
fontbold1));
dados.add(new Paragraph("Tempo da MIL Acesa: "
+ finalMilStamp + " min", fontbold1));
dados.add(new Paragraph("Comentrios Adicionais: "
+ comentxt, fontbold1));
pdfReport.add(dados);

pdfReport.add(esp);

pdfReport.add(addTitle2());

Paragraph in = new Paragraph();
in.add(new Paragraph(String.valueOf(failStamp),
fontbold1));
```

```
pdfReport.add(in);

pdfReport.newPage();
pdfReport.add(addTitle3());

if (paths[0] != null || paths[1] != null || paths[2]
!= null || paths[3] != null || paths[4] != null ||
paths[5] != null){

    PdfPTable tabimg = new PdfPTable(2);
    if (paths[0] != null) {
        Image img1 = Image.getInstance(paths[0]);
        tabimg.addCell(img1);
    }
    if (paths[1] != null) {
        Image img2 = Image.getInstance(paths[1]);
        tabimg.addCell(img2);
    }
    if (paths[2] != null) {
        Image img3 = Image.getInstance(paths[2]);
        tabimg.addCell(img3);
    }
    if (paths[3] != null) {
        Image img4 = Image.getInstance(paths[3]);
        tabimg.addCell(img4);
    }
    if (paths[4] != null) {
        Image img5 = Image.getInstance(paths[4]);
        tabimg.addCell(img5);
```

```
                }
                if (paths[5] != null) {
                    Image img6 = Image.getInstance(paths[5]);
                    tabimg.addCell(img6);
                }
                pdfReport.add(tabimg);
            }
        }catch(DocumentException | IOException de){
            Toast.makeText(getApplicationContext(),
        "Erro no Processo de Gerar o Relatrio!",
        Toast.LENGTH_LONG).show();
        } finally{
            pdfReport.close();
            Toast.makeText(getApplicationContext(),
            "Seu Relatrio Foi Gerado!", Toast.LENGTH_LONG)
            .show();
        }
        startActivity(report);
        finish();
    }
});


btncam = (Button) findViewById(R.id.btn_sim);
btncam.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent camera = new Intent(MediaStore.
        ACTION_IMAGE_CAPTURE);
```

```
                File photoFile = null;
                try{
                    photoFile = criarImagem();
                } catch (IOException e){
                    e.printStackTrace();
                }
                camera.putExtra(MediaStore.EXTRA_OUTPUT,
                 Uri.fromFile(photoFile));

                startActivityForResult(camera, CAM_REQUEST);

                for (int i=0; i<6; i++) {
                    setImageListener(i);
                }
            }
        });
    }


//--------Procedimentos Para Tirar Foto----------------//

    @Override
    protected void onActivityResult(int requestCode,
         int resultCode, Intent data) {
        if (requestCode == CAM_REQUEST &&
         resultCode == RESULT_OK) {

            int j = -1;
```

```
            for (int i=5; i>=0; i--) {
                if (paths[i] == null) {
                    j = i;
                }
            }
            if (j == -1) {
                return;
            }

            BitmapFactory.Options opt =
            new BitmapFactory.Options();
            opt.inSampleSize = 2;

            paths[j] = mLocalImagem;
            imageViews[j].setImageBitmap(
            BitmapFactory.decodeFile(mLocalImagem, opt));

            setImageListener(j);
        }
    }

    //---------Funo de Armazenamento da Imagem-----------//
    File criarImagem() throws IOException {

        String timeStamp = new SimpleDateFormat("ddMMyyyy_HHmmss")
        .format(new Date());
        String nomeDaImagem = "AdblueImg_" + timeStamp + "_";
        File storageDirectory = Environment.
        getExternalStoragePublicDirectory(
```

```
        Environment.DIRECTORY_PICTURES);

        File img = File.createTempFile(nomeDaImagem,
        ".jpg", storageDirectory);
        mLocalImagem = img.getAbsolutePath();

        return img;

    }

    //-----Redimensiona a Imagem e Volta Tambm seu Endereo----//
    public Bitmap getBitmapFromPath(String path) {
        BitmapFactory.Options opt = new BitmapFactory.Options();
        opt.inSampleSize = 2;

        return BitmapFactory.decodeFile(path, opt);
    }

    //----Funo Para Quando Desejar Editar As Imagens-------//
    public void setImageListener(int i) {
        if (paths[i] != null) {
            final int finalI = i;
            imageViews[i].setOnClickListener(
                new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    EditText placa = (EditText)
                    findViewById(R.id.insertxt_placa);
                    placatxt = placa.getText().toString();
```

```
                    EditText coment = (EditText)findViewById
                    (R.id.insertxt_coment);
                    comentxt = coment.getText().toString();
                    Intent imagem = new Intent(
                    InformationActivity.this, ImageActivity.class);
                    imagem.putExtra("flag", finalI);
                    imagem.putExtra("paths", paths);
                    imagem.putExtra("placatxt", placatxt);
                    imagem.putExtra("comentxt", comentxt);
                    imagem.putExtra("protStamp", protStamp);
                    imagem.putExtra("milStamp", milStamp);
                    imagem.putExtra("avinStamp", vinStamp);
                    imagem.putExtra("esnStamp", esnStamp);
                    imagem.putExtra("vehiStamp", vehiStamp);
                    imagem.putExtra("failStamp", failStamp);
                    imagem.putExtra("dateStamp", dateStamp);
                    imagem.putExtra("hourStamp", hourStamp);
                    startActivity(imagem);
                    finish();
                }
            });
        }
    }


//---------------Configuraodos Ttulos no PDF-----------------//

    public static Paragraph addTitle1(){
        Font fontbold = FontFactory.getFont(
```

```
        "Times-Roman", 20, Font.BOLD);
        Paragraph p = new Paragraph("INFORMAES GERAIS",
         fontbold);
        p.setSpacingAfter(20);
        p.setAlignment(1); // Center
        return p;
    }
    public static Paragraph addTitle2(){
        Font fontbold = FontFactory.getFont("Times-Roman",
         20, Font.BOLD);
        Paragraph p = new Paragraph("INCONSISTNCIAS",
 fontbold);
        p.setSpacingAfter(20);
        p.setAlignment(1); // Center
        return p;
    }
    public static Paragraph addTitle3(){
        Font fontbold = FontFactory.getFont("Times-Roman",
         20, Font.BOLD);
        Paragraph p = new Paragraph("FOTOS", fontbold);
        p.setSpacingAfter(20);
        p.setAlignment(1); // Center
        return p;
    }

}
```

APÊNDICE J

CLASSE - MAINACTIVITY

```
package com.antitamperingadblue;


        import android.Manifest;
        import android.app.Activity;
        import android.bluetooth.BluetoothAdapter;
        import android.bluetooth.BluetoothDevice;
        import android.bluetooth.BluetoothSocket;
        import android.content.Intent;
        import android.content.pm.PackageManager;
        import android.os.Bundle;
        import android.support.v4.app.ActivityCompat;
        import android.support.v7.app.AppCompatActivity;
        import android.view.View;
        import android.view.View.OnClickListener;
        import android.widget.ArrayAdapter;
        import android.widget.Button;
        import android.widget.Spinner;
        import android.widget.Toast;


        import java.io.IOException;
        import java.lang.reflect.InvocationTargetException;
        import java.util.Objects;
        import java.util.UUID;
```

```
public class MainActivity extends AppCompatActivity {

    //----------Declarao dos botes----------------------//
    Button btncall;
    Button btnsoft;
    Button btnnext;
    Button btntraining;
    Button btnconect;
    String mac = null;
    BluetoothAdapter bluetoothAdapter = null;
    BluetoothDevice bluetoothDevice = null;
    BluetoothSocket bluetoothSocket = null;
    Boolean  connected = false;
    private static final int ATIVAR_BLUETOOTH = 1;
    private static final int REQUEST_CONNECT_DEVICE = 2;
    private static final int REQUEST_BTCONNECT = 3;
    private static final int REQUEST_BTPAIRING = 4;
    UUID sppUUID = UUID.fromString(
        "00001101-0000-1000-8000-00805F9B34FB");

    //----------Inicializao do Layout----------------------//
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        final Spinner spinner;
        setContentView(R.layout.activity_main);
        spinner = (Spinner) findViewById(R.id.spinner);
        ArrayAdapter adapter = ArrayAdapter.createFromResource(this,
```

```
        R.array.lista_cam, android.R.layout.simple_spinner_item);
        spinner.setAdapter(adapter);

        bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

        if (bluetoothAdapter == null) {
            Toast.makeText(getApplicationContext(),
        "Seu Dispositivo No Possui Bluetooth",
        Toast.LENGTH_LONG).show();
        } else if (!bluetoothAdapter.isEnabled()) {
            Intent ativarBluetoothIntent = new Intent(
            BluetoothAdapter.ACTION_REQUEST_ENABLE);
            startActivityForResult(ativarBluetoothIntent,
                 ATIVAR_BLUETOOTH);
        }

//----------------Premisses Para Bluetooth---------------------//
        String[] PERMISSIONS_CONNECT = {
                Manifest.permission.BLUETOOTH,
                Manifest.permission.BLUETOOTH_ADMIN,
                Manifest.permission.BLUETOOTH_PRIVILEGED
        };
        int permission1 = ActivityCompat.checkSelfPermission(
        MainActivity.this, Manifest.permission.BLUETOOTH_ADMIN);
        if (permission1 != PackageManager.PERMISSION_GRANTED) {
            // Se no h permisso, promove o usurio
            ActivityCompat.requestPermissions(
                    MainActivity.this,
                    PERMISSIONS_CONNECT,
```

```
                REQUEST_BTCONNECT
        );
    }

    String[] PERMISSIONS_PAIRING = {
            Manifest.permission.ACCESS_COARSE_LOCATION,
            Manifest.permission.ACCESS_FINE_LOCATION
    };
    int permission2 = ActivityCompat.checkSelfPermission(
    MainActivity.this, Manifest.permission
    .ACCESS_COARSE_LOCATION);
    if (permission2 != PackageManager.PERMISSION_GRANTED) {
        // Se no h permisso, promove o usurio
        ActivityCompat.requestPermissions(
                MainActivity.this,
                PERMISSIONS_PAIRING,
                REQUEST_BTPAIRING
        );
    }

//------------------Botes--------------------------------//

    btnconect = (Button) findViewById(R.id.btn_conect);
    btnconect.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            if (connected) {
                try{
                    bluetoothSocket.close();
```

```
                connected = false;
                btnconect.setText(R.string.btn_conect);
                Toast.makeText(getApplicationContext(),
                "O Dispositivo Foi Desconectado!" ,
                Toast.LENGTH_SHORT).show();
            } catch (IOException e) {
                Toast.makeText(getApplicationContext(),
                "Ocorreu um erro " + e,
                Toast.LENGTH_SHORT).show();
            }
        }
        else {
            Intent abrirLista = new Intent(
            MainActivity.this, DeviceListActivity.class);
            startActivityForResult(abrirLista,
            REQUEST_CONNECT_DEVICE);
        }
    }
});

btncall = (Button) findViewById(R.id.btn_call);
btncall.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent callcenter = new Intent(MainActivity.this,
         CallActivity.class);
        startActivity(callcenter);
        finish();
    }
```

```
});

btnsoft = (Button) findViewById(R.id.btn_at);
btnsoft.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent software = new Intent(MainActivity.this,
        UpdateActivity.class);
        startActivity(software);
        finish();
    }
});

btnnext = (Button) findViewById(R.id.btn_ok);
btnnext.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        if (mac == null){
            Toast.makeText(getApplicationContext(),
            "Nenhum Dispositivo Conectado! Por Favor,
             Faa a Conexo Para Prosseguir" ,
                Toast.LENGTH_LONG).show();
        }
        else {
            try {
                bluetoothSocket.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
```

```
            String caminhao = spinner.getSelectedItem()
                .toString();

            if (Objects.equals(caminhao, "")){
                Toast.makeText(getApplicationContext(),
                 "Nenhum Caminho Selecionado! Por Favor,
                 Selecione o Moedelo Para Prosseguir" ,
                Toast.LENGTH_LONG).show();
            } else {
                Intent prox = new Intent(MainActivity.this,
                 VeriffActivity.class);
                prox.putExtra("caminhao", caminhao);
                prox.putExtra("address", mac);
                startActivity(prox);
                finish();
            }
        }
    }
});

btntraining = (Button) findViewById(R.id.btn_treino);
btntraining.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent treino = new Intent(MainActivity.this,
        TrainingActivity.class);
        startActivity(treino);
        finish();
    }
```

```
    });
}

@Override
protected void onActivityResult(int requestCode,
    int resultCode, Intent data) {
    switch(requestCode) {
        case ATIVAR_BLUETOOTH:
            if (resultCode == Activity.RESULT_OK) {
                Toast.makeText(getApplicationContext(),
                "Bluetooth Ativado!", Toast.LENGTH_LONG)
                .show();
            }else{
                Toast.makeText(getApplicationContext(),
                "Bluetooth No Ativado. O APP Foi Fechado!",
                 Toast.LENGTH_LONG).show();
                finish();
            }

            break;
        case REQUEST_CONNECT_DEVICE:
            if (resultCode == Activity.RESULT_OK)
            {
                Toast.makeText(getApplicationContext(),
                "Conectando...", Toast.LENGTH_LONG).show();
                final String address = data.getExtras().
                getString(DeviceListActivity.
                EXTRA_DEVICE_ADDRESS);
                Thread connectThread = new
```

```
                Thread(new Runnable() {
                    @Override
                    public void run() {
                        connectDevice(address);
                        mac = address;
                    }
                });
                connectThread.start();
            }
            break;
    }
}

private BluetoothSocket connectDevice(final
    String address) {
    bluetoothDevice = BluetoothAdapter.getDefaultAdapter().
    getRemoteDevice(address);
    try {
        bluetoothSocket = bluetoothDevice.
        createRfcommSocketToServiceRecord(sppUUID);
        bluetoothSocket = (BluetoothSocket) bluetoothDevice.
        getClass().getMethod("createRfcommSocket", new
        Class[] {int.class}).invoke(bluetoothDevice,1);
        bluetoothSocket.connect();
        this.runOnUiThread(new Runnable() {
            @Override
            public void run() {
                Toast.makeText(getApplicationContext(),
                "Conectado ao Dispositivo: " + address,
```

```
                    Toast.LENGTH_SHORT).show();
                    if (btnconect != null) btnconect.setText
                    (R.string.txt_desconectar);
                }
            });
            connected = true;
        } catch (final IOException erro) {
            connected = false;
            this.runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    Toast.makeText(getApplicationContext(),
                    "Ocorreu um erro " + erro,
                     Toast.LENGTH_SHORT).show();
                }
            });
        } catch (NoSuchMethodException |
            InvocationTargetException |
                 IllegalAccessException e) {
            e.printStackTrace();
        }
        return (bluetoothSocket);
    }
}
```

APÊNDICE K

CLASSE - MAINACTIVITYFRAGMENT

```
package com.antitamperingadblue;

        import android.os.Bundle;
        import android.support.v4.app.Fragment;
        import android.view.LayoutInflater;
        import android.view.View;
        import android.view.ViewGroup;
        import android.widget.TextView;


public class MainActivityFragment extends Fragment {

    public MainActivityFragment() {
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_main, container, false);
    }

    public void update(Integer id, String label) {
        View v = getView();
        if (v != null) {
```

```
            TextView tv = ((TextView) v.findViewById(id));
            if (tv != null) {
                tv.setText(label);
            }
        }
    }
}
```

APÊNDICE L

CLASSE - SHOWACTIVITY

```
package com.antitamperingadblue;

        import android.content.Intent;
        import android.graphics.Color;
        import android.os.Bundle;
        import android.support.v7.app.AppCompatActivity;
        import android.view.View;
        import android.widget.Button;
        import android.widget.TextView;

        import java.util.ArrayList;
        import java.util.Objects;


public class ShowActivity extends AppCompatActivity{

    //-----Declarao dos botes----------------------//
    Button btnnext;
    Button btnhome;
    Double NOX, TANK;
    Integer COM;
    String interruptStamp = null;
    String tanqueStamp = null;
    String nox3Stamp = null;
```

```
    String nox7Stamp = null;
    String falhaStamp = null;
    String plausStamp = null;
    ArrayList<String> failStamp;


    //--------------Inicializao do Layout---------------//
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.show_layout);

        failStamp = new ArrayList<>();

        Intent analise = getIntent();
//-----Recebe os Valores das Flags do Pacote Enviado da ChooseActivity----//
        int flag_interrupt = analise.getIntExtra("flag_interrupt", 0);
        int flag_tanque = analise.getIntExtra("flag_tanque", 0);
        int flag_nox3 = analise.getIntExtra("flag_nox3", 0);
        int flag_nox7 = analise.getIntExtra("flag_nox7", 0);
        int flag_falha = analise.getIntExtra("flag_falha", 0);
        int flag_plaus = analise.getIntExtra("flag_plaus", 0);
        String NOXS = analise.getStringExtra("NOXS");
        String TANKS = analise.getStringExtra("TANKS");
        String COMS = analise.getStringExtra("COMS");
        String intElet = analise.getStringExtra("intElet");
        final String protStamp = analise.getStringExtra("protStamp");
        final String milStamp = analise.getStringExtra("milStamp");
        final String vinStamp = analise.getStringExtra("vinStamp");
```

```
        final String esnStamp = analise.getStringExtra("esnStamp");
        final String vehiStamp = analise.getStringExtra("vehiStamp");
        final String TANKF = analise.getStringExtra("TANKF");
        final String NOX3F = analise.getStringExtra("NOX3F");
        final String NOX7F = analise.getStringExtra("NOX7F");
        final String COMF = analise.getStringExtra("COMF");
        final String PLAUF = analise.getStringExtra("PLAUF");

        if (!Objects.equals(NOXS, "N/A")) {
            NOX = Double.valueOf(NOXS);
        }
        if(!Objects.equals(TANKS, "N/A")){
            TANK = Double.valueOf(TANKS);
        }
        if (!Objects.equals(NOXS, "N/A")) {
            COM = Integer.parseInt(COMS);
        }

//----Atribui aos radiobuttons os ids de seus equivalentes no layout---------//
        TextView tanque = (TextView) findViewById(R.id.check_interrup);
        TextView nox3 = (TextView) findViewById(R.id.check_tanque);
        TextView nox7 = (TextView) findViewById(R.id.check_nox3_5);
        TextView falha = (TextView) findViewById(R.id.check_nox7_0);
        TextView interrupt = (TextView) findViewById(R.id.check_falha);
        TextView plaus = (TextView) findViewById(R.id.check_plau);


//-Exibir ou Ocultar a Medio em Questo Caso No Tenha Sido Selecionada--//
        if (flag_interrupt == 1) {
```

```
        if (Objects.equals(intElet, "N/A")){
            interrupt.setVisibility(View.VISIBLE);
            interrupt.setText(intElet);
            interrupt.setTextColor(Color.WHITE);
        } else {
            if (intElet.equals("Normal")) {
                interrupt.setVisibility(View.VISIBLE);
                interrupt.setTextColor(Color.GREEN);
            }
            if (intElet.equals("Falha")) {
                interrupt.setVisibility(View.VISIBLE);
                interrupt.setTextColor(Color.RED);
                interruptStamp = (String) interrupt.getText();
                failStamp.add(interruptStamp);
            }
        }
    }

    if (flag_interrupt == 0){
        if (Objects.equals(intElet, "N/A")){
            interrupt.setVisibility(View.INVISIBLE);
        } else {
            if (intElet.equals("Normal")) {
                interrupt.setVisibility(View.INVISIBLE);
            }
            if (intElet.equals("Falha")) {
                interrupt.setVisibility(View.VISIBLE);
                interrupt.setTextColor(Color.RED);
                interruptStamp = (String) interrupt.getText();
```

```
                failStamp.add(interruptStamp);
            }
        }
    }

    if (flag_tanque == 1){
        if (Objects.equals(TANKF, "N/A")){
            tanque.setVisibility(View.VISIBLE);
            tanque.setText(TANKF);
            tanque.setTextColor(Color.WHITE);
        } else {
            if (TANKF.equals("Normal")) {
                tanque.setVisibility(View.VISIBLE);
                tanque.setTextColor(Color.GREEN);
            }

            if (TANKF.equals("Falha")) {
                tanque.setVisibility(View.VISIBLE);
                tanque.setTextColor(Color.RED);
                tanqueStamp = (String) tanque.getText();
                failStamp.add(tanqueStamp);
            }
        }
    }

    if (flag_tanque == 0){
        if (Objects.equals(TANKF, "N/A")){
            tanque.setVisibility(View.INVISIBLE);
        } else {
```

```
            if (TANKF.equals("Normal")) {
                tanque.setVisibility(View.INVISIBLE);
            }
            if (TANKF.equals("Falha")) {
                tanque.setVisibility(View.VISIBLE);
                tanque.setTextColor(Color.RED);
                tanqueStamp = (String) tanque.getText();
                failStamp.add(tanqueStamp);
            }
        }
    }

    if (flag_nox3 == 1) {
        if (Objects.equals(NOX3F, "N/A")){
            nox3.setVisibility(View.VISIBLE);
            nox3.setText(NOX3F);
            nox3.setTextColor(Color.WHITE);
        } else {

            if (NOX3F.equals("Normal")) {
                nox3.setVisibility(View.VISIBLE);
                nox3.setTextColor(Color.GREEN);
            }
            if (NOX3F.equals("Falha")) {
                nox3.setVisibility(View.VISIBLE);
                nox3.setTextColor(Color.RED);
                nox3Stamp = (String) nox3.getText();
                failStamp.add(nox3Stamp);
            }
```

```
    }
}

if (flag_nox3== 0){
    if (Objects.equals(NOX3F, "N/A")){
        nox3.setVisibility(View.INVISIBLE);
    } else {
        if (NOX3F.equals("Normal")) {
            nox3.setVisibility(View.INVISIBLE);
        }
        if (NOX3F.equals("Falha")) {
            nox3.setVisibility(View.VISIBLE);
            nox3.setTextColor(Color.RED);
            nox3Stamp = (String) nox3.getText();
            failStamp.add(nox3Stamp);
        }
    }
}

if (flag_nox7 == 1){
    if (Objects.equals(NOX7F, "N/A")){
        nox7.setVisibility(View.VISIBLE);
        nox7.setText(NOX7F);
        nox7.setTextColor(Color.WHITE);
    } else {
        if (NOX7F.equals("Normal")) {
            nox7.setVisibility(View.VISIBLE);
            nox7.setTextColor(Color.GREEN);
        }
```

```
            if (NOX7F.equals("Falha")) {
                nox7.setVisibility(View.VISIBLE);
                nox7.setTextColor(Color.RED);
                nox7Stamp = (String) nox7.getText();
                failStamp.add(nox7Stamp);
            }
        }
    }

    if (flag_nox7 == 0){
        if (Objects.equals(NOX7F, "N/A")){
            nox7.setVisibility(View.INVISIBLE);
        } else {
            if (NOX7F.equals("Normal")) {
                nox7.setVisibility(View.INVISIBLE);
            }
            if (NOX7F.equals("Falha")) {
                nox7.setVisibility(View.VISIBLE);
                nox7.setTextColor(Color.RED);
                nox7Stamp = (String) nox7.getText();
                failStamp.add(nox7Stamp);
            }
        }
    }

    if (flag_falha == 1){
        if (Objects.equals(COMF, "N/A")){
            falha.setVisibility(View.VISIBLE);
            falha.setText(COMF);
```

```
            falha.setTextColor(Color.WHITE);
        } else {
            if (COMF.equals("Normal")) {
                falha.setVisibility(View.VISIBLE);
                falha.setTextColor(Color.GREEN);
            }
            if (COMF.equals("Falha")) {
                falha.setVisibility(View.VISIBLE);
                falha.setTextColor(Color.YELLOW);
                falhaStamp = (String) falha.getText();
                failStamp.add(falhaStamp);
            }
        }
    }

    if (flag_falha == 0){
        if (Objects.equals(COMF, "N/A")){
            falha.setVisibility(View.INVISIBLE);
        } else {
            if (COMF.equals("Normal")) {
                falha.setVisibility(View.INVISIBLE);
            }
            if (COMF.equals("Falha")) {
                falha.setVisibility(View.VISIBLE);
                falha.setTextColor(Color.YELLOW);
                falhaStamp = (String) falha.getText();
                failStamp.add(falhaStamp);
            }
        }
```

```
}

if (flag_plaus == 1){
    if (Objects.equals(PLAUF, "N/A")){
        plaus.setVisibility(View.VISIBLE);
        plaus.setText(PLAUF);
        plaus.setTextColor(Color.WHITE);
    } else {
        if (PLAUF.equals("Normal")) {
            plaus.setVisibility(View.VISIBLE);
            plaus.setTextColor(Color.GREEN);
        }
        if (PLAUF.equals("Falha")) {
            plaus.setVisibility(View.VISIBLE);
            plaus.setTextColor(Color.YELLOW);
            plausStamp = (String) plaus.getText();
            failStamp.add(plausStamp);
        }
    }
}

if (flag_plaus == 0){
    if (Objects.equals(PLAUF, "N/A")){
        plaus.setVisibility(View.INVISIBLE);
    } else {
        if (PLAUF.equals("Normal")) {
            plaus.setVisibility(View.INVISIBLE);
        }
        if (PLAUF.equals("Falha")) {
```

```
                plaus.setVisibility(View.VISIBLE);
                plaus.setTextColor(Color.YELLOW);
                plausStamp = (String) plaus.getText();
                failStamp.add(plausStamp);
            }
        }
    }

    btnnext = (Button) findViewById(R.id.btn_report);
    btnnext.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent report = new Intent(ShowActivity.this,
            InformationActivity.class);
            report.putExtra("protStamp", protStamp);
            report.putExtra("milStamp", milStamp);
            report.putExtra("avinStamp", vinStamp);
            report.putExtra("esnStamp", esnStamp);
            report.putExtra("vehiStamp", vehiStamp);
            report.putExtra("failStamp",failStamp);
            startActivity(report);
            finish();
        }
    });

    btnhome = (Button) findViewById(R.id.btn_volt4);
    btnhome.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
```

```
                Intent home = new Intent(ShowActivity.this, MainActivity.class);
                startActivity(home);
                finish();
            }
        });
    }
}
```

APÊNDICE M

CLASSE - SPLASHACTIVITY

```
package com.antitamperingadblue;



        import android.content.Intent;
        import android.os.Bundle;
        import android.os.Handler;
        import android.support.v7.app.AppCompatActivity;
        import android.util.Log;
        import android.widget.ProgressBar;



public class SplashActivity extends AppCompatActivity{

    //-------------Declarao dos Utilitrios--------------------//
    private final static int TIME_SPLASH = 3000;
    protected boolean nbActive;
    protected ProgressBar nProgressBar;

    //--------------Inicializao do Layout----------------------//
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.splash_layout);
```

```
        nProgressBar = (ProgressBar)findViewById(R.id.pb_splash);

//-----------------Thread de Contagem de Tempo-------------------//
        final Thread timerThread = new Thread() {
            @Override
            public void run() {
                nbActive = true;
                try{
                    int waited = 0;
                    while (nbActive && (waited < TIME_SPLASH)) {
                        sleep(50);
                        if(nbActive) {
                            waited += 50;
                            updateProgress(waited);
                        }
                    }
                } catch(InterruptedException e) {
                    // caso erro!!
                } finally {
                    onContinue();
                }
            }
        };
        timerThread.start();

        new Handler().postDelayed(new Runnable() {
            @Override
            public void run() {
                Intent dashboard = new Intent (SplashActivity.this,
```

```
                MainActivity.class);
                startActivity(dashboard);
                finish();
            }
        },TIME_SPLASH);
    }

    //-------Funo Para Atualizar o Progresso da Barra--------//
    public void updateProgress (final int timePassed) {
        if (null != nProgressBar) {

            final int progress = nProgressBar.getMax() *timePassed / TIME_SPLASH;
            nProgressBar.setProgress(progress);

        }
    }

    public void onContinue() {
        Log.d("mensagemFinal", "Sua barra acabou de carregar!!!");
    }

}
```

APÊNDICE N

CLASSE - TRAININGACTIVITY

```
package com.antitamperingadblue;


        import android.content.Intent;
        import android.os.Bundle;
        import android.support.v7.app.AppCompatActivity;
        import android.view.View;
        import android.widget.Button;



public class TrainingActivity extends AppCompatActivity {

    //------------Declarao dos botes-------------------//
    Button btnhome;

    //------Inicializao do Layout----------------------//
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.training_layout);

        btnhome = (Button) findViewById(R.id.btn_ok4);
        btnhome.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
```

```
                Intent home = new Intent(TrainingActivity.this,
                MainActivity.class);
                startActivity(home);
                finish();
            }
        });
    }
}
```

APÊNDICE O

CLASSE - UPDATEACTIVITY

```
package com.antitamperingadblue;

import android.content.Intent;
        import android.os.Bundle;
        import android.support.v7.app.AppCompatActivity;
        import android.view.View;
        import android.widget.Button;


public class UpdateActivity extends AppCompatActivity{

    //-------------Declarao dos botes-------------------//
    Button btnhome;

    //--------Inicializao do Layout-------------------//
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.update_layout);

        btnhome = (Button) findViewById(R.id.btn_volt3);
        btnhome.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
```

```
                Intent home  = new Intent(UpdateActivity.this,
                 MainActivity.class);
                startActivity(home);
                finish();
            }
        });
    }
}
```

APÊNDICE P

CLASSE - VERIFFACTIVITY

```
package com.antitamperingadblue;



import android.bluetooth.BluetoothAdapter;
        import android.bluetooth.BluetoothDevice;
        import android.bluetooth.BluetoothSocket;
        import android.content.Intent;
        import android.os.Bundle;
        import android.support.v4.app.Fragment;
        import android.support.v7.app.AppCompatActivity;
        import android.util.Log;
        import android.view.View;
        import android.widget.Button;
        import android.widget.Toast;

        import java.io.IOException;
        import java.io.InputStream;
        import java.lang.reflect.InvocationTargetException;
        import java.util.HashMap;
        import java.util.Map;
        import java.util.UUID;



public class VeriffActivity extends AppCompatActivity {
```

```
//-----------Declarao dos Botes e Utilitrios-----------//
Button btnnext;
double NOX, TANK;
int COM, flag, FMI, SPN;
String MIL, MILS, FMIS, TANKF, PLAUF, COMF, NOX3F, NOX7F, intElet, mac,
       SPNS, VIN, ESN, COMS, NOXS, TANKS, ISO = "ISO 15031",
        SAE = "SAE J1939",
        SAE2 = "J1939/1587", SAISO = "SAE J1939/1587 & ISO 15031";
BluetoothDevice bluetoothDevice = null;
BluetoothSocket bluetoothSocket = null;
UUID sppUUID = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");
private static final String TAG = "AdBlue";
private static final byte RS232_FLAG = (byte) 0xC0;
private static final byte RS232_ESCAPE = (byte) 0xDB;
private static final byte RS232_ESCAPE_FLAG = (byte) 0xDC;
private static final byte RS232_ESCAPE_ESCAPE = (byte) 0xDD;
private static final int FA_J1939 = 1;
private static final int FD_J1939 = 2;
private static final int TX_J1939 = 5;
private static final int RX_J1939 = 6;
private static final int TX_OBD = 42;
private static final int STATS = 23;
private HashMap<String, String> newData;
private HashMap<String, Integer> monitorFields;
String vehiStamp;
private boolean isInvalid;
private boolean isStuffed;
private int m_size;
```

```
    private int m_count;
    private byte[] m_buffer;


    //---------Inicializao do Layout-----------------------------//
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.layout_activity);

//-----Adquirir Dados Enviados Pela Intent da Main Activity-------//
        Intent caminhaoIntent = getIntent();
        vehiStamp = caminhaoIntent.getStringExtra("caminhao");
        mac = caminhaoIntent.getStringExtra("address");


/-Atribuio Dos Protocolos Respectivos Aos Modelos dos Caminhes--//

        if (vehiStamp.contentEquals("Iveco Stralis")) {
            flag = 1;
        } else if (vehiStamp.contentEquals("Iveco Tector")) {
            flag = 1;
        } else if (vehiStamp.contentEquals("MAN Delivery")) {
            flag = 2;
        } else if (vehiStamp.contentEquals("MAN Constellation 17/19/24.33")) {
            flag = 2;
        } else if (vehiStamp.contentEquals("MAN TGX 440/480")) {
            flag = 1;
        } else if (vehiStamp.contentEquals("Mercedes Benz (Todos)")) {
```

```
    flag = 1;
} else if (vehiStamp.contentEquals("Scania (Todos)")) {
    flag = 1;
} else if (vehiStamp.contentEquals("VOLVO VM")) {
    flag = 3;
} else if (vehiStamp.contentEquals("VOLVO FH/FM")) {
    flag = 4;
}

initTextViews(flag);

Thread connectThread = new Thread(new Runnable() {
    @Override
    public void run() {
        connectDevice(mac);
    }
});

connectThread.start();


btnnext = (Button) findViewById(R.id.btn_Ok);
btnnext.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        try {
            bluetoothSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
```

```
}
Intent escolha = new Intent(VeriffActivity.this,
ChooseActivity.class);
String milStamp = MILS;
String vinStamp = VIN;
String esnStamp = ESN;
String protStamp = null;
if (flag == 1){protStamp = ISO;}
if (flag == 2){protStamp = SAE;}
if (flag == 3){protStamp = SAE2;}
if (flag == 4){protStamp = SAISO;}
if (NOXS == null ){
    NOXS = "N/A";
}
if (TANKS == null){
    TANKS = "N/A";
}
if (COMS == null){
    COMS = "N/A";
}
if (intElet == null){
    intElet = "N/A";
}
if (TANKF == null){
    TANKF = "N/A";
}
if (NOX3F == null){
    NOX3F = "N/A";
}
```

```
            if (NOX7F == null){
                NOX7F = "N/A";
            }
            if (COMF == null){
                COMF = "N/A";
            }
            if (PLAUF == null){
                PLAUF = "N/A";
            }
            escolha.putExtra("protStamp", protStamp);
            escolha.putExtra("milStamp", milStamp);
            escolha.putExtra("vinStamp", vinStamp);
            escolha.putExtra("esnStamp", esnStamp);
            escolha.putExtra("vehiStamp", vehiStamp);
            escolha.putExtra("intElet", intElet);
            escolha.putExtra("NOXS", NOXS);
            escolha.putExtra("COMS", COMS);
            escolha.putExtra("TANKS", TANKS);
            escolha.putExtra("TANKF", TANKF);
            escolha.putExtra("NOX3F", NOX3F);
            escolha.putExtra("NOX7F", NOX7F);
            escolha.putExtra("COMF", COMF);
            escolha.putExtra("PLAUF", PLAUF);
            startActivity(escolha);
            finish();
        }
    });

}
```

```
private void initTextViews(int flag) {
    newData = new HashMap<>();
    monitorFields = new HashMap<>();
    newData.put("Protocolo", "");
    monitorFields.put("Protocolo", R.id.ProtField);
    if (flag == 1) {
        newData.put("Protocolo", ISO);
    } else if (flag == 2) {
        newData.put("Protocolo", SAE);
    } else if (flag == 3) {
        newData.put("Protocolo", SAE2);
    } else if (flag == 4)
    {
        newData.put("Protocolo", SAISO);
    }
    newData.put("Tempo da MIL Acesa", "");
    monitorFields.put("Tempo da MIL Acesa",
    R.id.MILField);
    newData.put("VIN", "");
    monitorFields.put("VIN", R.id.VINField);
    newData.put("Nmero de Srie do Motor","");
    monitorFields.put("Nmero de Srie do Motor", R.id
    .ESNField);
    newData.put("DTC1","");
    monitorFields.put("DTC1", R.id .DTC1Field);
    newData.put("DTC2","");
    monitorFields.put("DTC2", R.id .DTC2Field);
    newData.put("DTC3","");
```

```
        monitorFields.put("DTC3", R.id .DTC3Field);
        newData.put("DTC4","");
        monitorFields.put("DTC4", R.id .DTC4Field);

    }

//---Thread de Leitura Dos Dsdos Recebidos Pelo Disp. BT----//

    private final Runnable readRun = new Runnable() {
        public void run() {
            receiveDataFromBT(bluetoothSocket);
        }
    };
    private Thread readThread;

//--Funo Para Fazer a Conexo Com o Disp. BT----------//

    private BluetoothSocket connectDevice(final String address) {
        bluetoothDevice = BluetoothAdapter.getDefaultAdapter().
        getRemoteDevice(address);
        try {
            bluetoothSocket = bluetoothDevice.
            createRfcommSocketToServiceRecord(sppUUID);
            bluetoothSocket = (BluetoothSocket) bluetoothDevice.
            getClass().getMethod("createRfcommSocket", new Class[]
            {int.class}).invoke(bluetoothDevice, 1);
            bluetoothSocket.connect();

            init_j1939();
```

```
            read();

        } catch (final IOException erro) {
            this.runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    Toast.makeText(getApplicationContext(),
                    "Ocorreu um erro " + erro, Toast.LENGTH_SHORT)
                    .show();
                }
            });
        } catch (NoSuchMethodException | InvocationTargetException |
                 IllegalAccessException e) {
            e.printStackTrace();
        }
        return (bluetoothSocket);
    }

    private void read(){
        if (readThread != null && readThread.isAlive()) {
            readThread.interrupt();
            while (readThread.isAlive()) Thread.yield();
        } else {
            // Se no houver nada na thread de leitura inicia uma nova
            readThread = new Thread(readRun);
            readThread.setPriority(4);
            readThread.start();
        }
```

```
    }

    //--------Funo Receber Os Dados Do Disp. BT------------------//
    private void receiveDataFromBT(BluetoothSocket socket) {
        try {
            byte[] buffer = new byte[1024];
            int buf_len = 0;

            if (socket == null) {
                return;
            }

            InputStream inputStream = socket.getInputStream();

            while (true) {
                try {
  // Se a Thread for interrompida, encerra a leitura do InputStream
                    if (Thread.interrupted()) {
                        assert inputStream != null;
                        inputStream.close();
                        return;
                    }
                    // Receber dados da InputStream
                    if (inputStream != null) {
                        buf_len = inputStream.read(buffer);
                    }
                    Thread.sleep(1);
  // Se o valor do tamanho da msg do buffer for -1 encerra a leitura
                    if (buf_len == -1) {
```

```
                    inputStream.close();
                    break;
                }

                parseMessage(buffer, buf_len);

            } catch (IOException e) {
                if (Thread.interrupted()) {
                    inputStream.close();
                    return;
                }
                break;
            } catch (InterruptedException e) {
                assert inputStream != null;
                inputStream.close();
                Log.e(TAG, "Interrupted read", e);
                return;
            }
        }

    } catch (IOException e) {
        Log.e(TAG, "", e);
    }
}

//--Funo Para Analisar A Mensagem de Cada Posio do Buffer--//
private void parseMessage(byte[] buf, int len) {
    for (int i = 0; i < len; i++) {
        processCharFromBus(buf[i]);
```

```
    }
}

//--Verifica Se o Tamanho t de Acordo Com o Inicio do Pacote--/
private void processCharFromBus(byte val) {
    try {
        // o comeo da mensagem?
        if (val == RS232_FLAG) {
            isInvalid = false;
            isStuffed = false;
            m_size = -1;
            m_count = 0;
        } else if (!isInvalid) {
            if (val == RS232_ESCAPE) {
                isStuffed = true;
            } else {
        //Se o byte anterior foi preenchido, decodifica esse byte
                if (isStuffed) {
                    isStuffed = false;
                    if (val == RS232_ESCAPE_FLAG) {
                        val = RS232_FLAG;
                    } else if (val == RS232_ESCAPE_ESCAPE) {
                        val = RS232_ESCAPE;
                    } else {
                        isInvalid = true;
      //Byte invlido depois de preencher, aborta o processo
                        return;
                    }
                }
```

```
                if (m_count < m_buffer.length) {
                    m_buffer[m_count] = val;
                    m_count++;
                }

    // Quando obtivermos 2 bytes, teremos informaes o
    // bastante pra calcular o verdadeiro tamamho da mensagem
                if (m_count == 2) {
                    m_size = ((m_buffer[0] << 8) | m_buffer[1]) + 2;
                }

      //A mensagem foi recebida por completo? Se foi,  vlida?
                if (m_count == m_size && val == cksum(m_buffer,
                    m_count - 1)) {
                    m_count--;
                    processPacket(m_buffer);
                }
            }
        }
    } catch (Exception e) {
        System.out.println(e.getStackTrace()[0]);
    }
}

//---Processar o Pacote e Adquirir As Mensagens Desejadas--//
private void processPacket(byte[] packet) {
    // c0 00 0a 05 00 pp gg nn 00 00 00 ff xx
    //                   PGN
    Integer i;
```

```
int msgID = packet[2];
int tam = (packet[1] - 11)/4;
if (msgID == RX_J1939) {
    final Integer pgn = ((packet[4] & 0xFF) << 16) |
        ((packet[5] & 0xFF) << 8) | (packet[6] & 0xFF);
    // Pacote [9 Bits Header + o Bit De Posio Do SPN]
    switch (pgn) {
        case XXXXX:
            MIL = String.valueOf(((packet[15] & 0xFF)
                << 8) | (packet[14] & 0xFF));
            MILS = MIL + " min";
            newData.put("Tempo da MIL Acesa", MILS);
            break;
       /* case XXXXXX:
            NOXS = String.valueOf(((packet[11] & 0xFF)
                << 8) | (packet[10] & 0xFF));
            NOX = Integer.parseInt(NOXS);
            NOX = (NOX * 0.05) - 200;
            NOXS = String.valueOf(NOX);
            COMS = String.valueOf((packet[14] & 0x03));
            COM = Integer.parseInt(COMS);
            break;
         case XXXXXXX:
            TANKS = String.valueOf(packet[10] & 0xFF);
            TANK = Integer.parseInt(TANKS);
            TANK = (TANK / 255) * 100;
            TANKS = String.valueOf(TANK);
            break;*/
        case XXXXXXXXX:
```

```
            VIN = String.valueOf(((packet[10] & 0xFF)
                <<8)|(packet[11] & 0xFF));
            newData.put("VIN", VIN);
            break;
        case XXXXXXXX:
            ESN = String.valueOf((packet[10] & 0xFF));
            newData.put("Nmero de Srie do Motor", ESN);
            break;
        case XXXXXXXXX:
            intElet = "Normal";
            TANKF = "Normal";
            NOX3F = "Normal";
            NOX7F = "Normal";
            COMF = "Normal";
            PLAUF = "Normal";
            for(i=0;i<tam;i++) {
                Integer bt = (packet[14 + i*4] & 0xFF);
                Integer btt = (bt & 0x1F);
                SPNS = String.valueOf(((bt & 0xE0)
                << 11) | ((packet[13 + i * 4] & 0xFF)
                << 8) | (packet[12 + i * 4] & 0xFF));
                SPN = Integer.parseInt(SPNS);
                if(i==0){
                    newData.put("DTC1", "SPN: "+ SPNS
                        + "   " + "FMI: " + btt);
                }
                if(i==1){
                    newData.put("DTC2", "SPN: "+ SPNS
                        + "   " + "FMI: " + btt);
```

```
}
if(i==2){
    newData.put("DTC3", "SPN: "+ SPNS
        + "   " + "FMI: " + btt);
}
if(i==3){
    newData.put("DTC4", "SPN: "+ SPNS
        + "   " + "FMI: " + btt);
}
if (SPN == XXXXX) {
    FMIS = String.valueOf(bt & 0x1F);
    FMI = Integer.parseInt(FMIS);
    if (FMI == XX || FMI == X) {
        intElet = "Falha";
    }
}

if (SPN == XXX) {
    FMIS = String.valueOf(bt & 0x1F);
    FMI = Integer.parseInt(FMIS);
    if (FMI == X || FMI == XXX) {
        TANKF = "Falha";
    }
}

if (SPN == XXXX) {
    FMIS = String.valueOf(bt & 0x1F);
    FMI = Integer.parseInt(FMIS);
    if (FMI == XX) {
```

```
                        NOX3F = "Falha";
                    }
                    if (FMI == XXX){
                        NOX7F = "Falha";
                    }
                    if (FMI == XXX){
                        COMF = "Falha";
                    }
                    if (FMI == XXXX){
                        COMF = "Falha";
                        PLAUF = "Falha";
                    }
                }

                if (SPN == XXXXX) {
                    FMIS = String.valueOf(bt & 0x1F);
                    FMI = Integer.parseInt(FMIS);
                    if (FMI == XXX) {
                        COMF = "Falha";
                    }
                    if (FMI == XXX){
                        PLAUF = "Falha";
                    }
                }
            }
            break;

        }
    } else if (msgID == STATS) {
```

```
            this.runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    updateLabels();
                }
            });

        }
    }

    private void updateLabels() {
        for(Fragment f : getSupportFragmentManager().
                getFragments()) {
            if(f != null && f.getClass().equals(
                        MainActivityFragment.class)) {
                for (Map.Entry<String, Integer> entry
                        : monitorFields.entrySet()) {
                    Integer tv = entry.getValue();
                    String label = newData.get(entry.
                        getKey());
                    if (!label.equals("")) {
                        if (tv != null) {
                            ((MainActivityFragment)f).
                        update(tv, label);
                        }
                    }
                }
            }
        }
```

```
}


//-Enviar a Requisio Dos Dados Dos Respectivos PGNs----//
private void init_j1939()
{
    m_buffer = new byte[4096];
    m_count = 0;


    // Seleciona os PGNs solicitados
    long[] initPGN_AddFilter1 = {XXXXX,XXXXX,XXXXX};
    long[] initPGN_AddFilter = {XXXXXX, XXXXX, XXXX};

    for(long pgn:initPGN_AddFilter1)
    {
        sendCommand(filterAddDelJ1939((byte) 0, pgn, true));
    }

    for(long pgn:initPGN_AddFilter)
    {
        sendCommand(filterAddDelJ1939((byte) 0, pgn, true));
    }
    sendCommand(requestJ1939((byte) 0, XXXX));
    try {
        Thread.sleep(1000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
```

```
    sendCommand(requestJ1939((byte) 0, XXXXX));
    try {
        Thread.sleep(1000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    sendCommand(requestJ1939((byte) 0, XXXXX));
}
//-----Enviar Os Comandos de Requisio De Dados--//
private void sendCommand(TxStruct command)
{
    if (bluetoothSocket != null)
    {
        try
        {
            bluetoothSocket.getOutputStream().write
            (command.getBuf(),0,command.getLen());
        }
        catch (IOException e)
        {
            Log.e(TAG, "Send Command Socket Closed", e);
            disconnect();
            connectDevice(mac);
        }
    }
    else
    {
        disconnect();
    }
```

```
    }

//----Executar o Checksum da Msg Recebida-----//
    private int cksum(byte[] data, int numbytes) {
        int count = 0;

        for (int i = 0; i < numbytes; i++) {
            count += uByte(data[i]);
        }
        return (byte) (~(count & 0xFF) + (byte) 1);
    }

    private int uByte(byte b)
    {
        return (int)b & 0xFF;
    }



    //------Funo Para os Parmetros de Transmisso-------//
    private class TxStruct {
        private byte[] buf;
        private int len;

        TxStruct(byte[] buf, int len) {
            this.buf = buf;
            this.len = len;
        }
        int getLen() {
```

```
            return len;
        }
        byte[] getBuf() {
            return buf;
        }
    }
}
```