BRUNO CÉSAR FERNANDES PEREIRA

**Evolução de uma Unidade de Gerenciamento Eletrônico
de um Motor VW 2.0L e Desenvolvimento de Controle
de Cruzeiro: Projeto Otto IV**

São Paulo

2017

BRUNO CÉSAR FERNANDES PEREIRA

**Evolução de uma Unidade de Gerenciamento Eletrônico
de um Motor VW 2.0L e Desenvolvimento de Controle
de Cruzeiro: Projeto Otto IV**

Dissertação apresentada à Escola Politécnica
da Universidade de São Paulo para a obtenção
do Título de Mestre em Ciências.

São Paulo

2017

BRUNO CÉSAR FERNANDES PEREIRA

Evolução de uma Unidade de Gerenciamento Eletrônico de um Motor VW 2.0L e Desenvolvimento de Controle de Cruzeiro: Projeto Otto IV

Dissertação apresentada à Escola Politécnica
da Universidade de São Paulo para a obtenção
do Título de Mestre em Ciências.

Área de Concentração:
Engenharia Elétrica - Sistemas Eletrônicos

Orientador: Prof. Dr. Armando Antonio Ma-
ria Laganá

São Paulo

2017

Este exemplar foi revisado e corrigido em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, _____ de _____ de _____

Assinatura do autor: _____

Assinatura do orientador: _____

Catálogo-na-publicação

Pereira, Bruno César Fernandes

Evolução de uma Unidade de Gerenciamento Eletrônico de um Motor VW 2.0L e Desenvolvimento de Controle de Cruzeiro: Projeto Otto IV / B. C. F. Pereira -- versão corr. -- São Paulo, 2017.
274 p.

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Sistemas Eletrônicos.

1.Eletrônica Embarcada 2.Motores Ciclo Otto 3.Engenharia Automotiva 4.Sistemas de Controle 5.Sistemas Veiculares I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Sistemas Eletrônicos II.t.

Agradecimentos

Primeiramente um agradecimento em especial e com muito amor aos meus queridos pais, Maria da Conceição Fernandes Pereira e Nerci Florêncio Pereira, por todo amor, dedicação, e apoio cedidos durante uma vida inteira. São verdadeiros exemplos de vida e espero um dia retribuir tudo o que fizeram por mim.

À minha namorada, Josiane Luz Pires, por estar sempre ao meu lado, prestando constante apoio e incentivo ao decorrer deste trabalho, sempre com muito amor e carinho. Obrigado por ser essa companheira de vida tão especial.

Ao professor e amigo doutor Armando Antonio Maria Laganá, orientador e idealizador deste projeto, por uma dedicação única que certamente extrapolou as obrigações de um orientador acadêmico. O sucesso deste trabalho está diretamente atrelado ao seu imenso apoio, despejando confiança ao crer no potencial de um aluno, além de fornecer todo o suporte necessário para a concretização do projeto. Certamente será sempre um grande exemplo de pessoa a ser seguido.

Aos grandes amigos da FATEC Santo André, Demerson Moscardini e Bruno Silva Pereira, por todo o apoio, esforço e dedicação prestados ao longo da parte prática deste projeto, fato decisivo para o sucesso deste trabalho. Foi um prazer imenso trabalhar em uma equipe formada por pessoas tão dedicadas e talentosas. Desejo-lhes muito sucesso em suas vidas acadêmicas e profissionais.

Ao professor Francisco Antonio Marino Salotti, que cedeu e prezou pela manutenção da excelente estrutura presente no Instituto de Energia e Ambiente da USP para testes com veículos automotivos. Esta estrutura conta com um moderno dinamômetro inercial que, por sua vez, foi definitivo para o sucesso de diversos testes realizados ao longo do trabalho.

Ao professor doutor Bruno Augusto Angélico, pelas dicas fornecidas para a modelagem do controlador de cruzeiro desenvolvido neste trabalho.

Ao professor doutor Eduardo Lorenzetti Pellini que, conjuntamente ao professor doutor Armando Antonio Maria Laganá, dedicou parte de seu tempo à revisão desta obra, fornecendo dicas e orientações em relação ao trabalho.

À FATEC Santo André, pelo apoio dado cedendo o veículo para testes.

Por fim, a todos que colaboraram direta ou indiretamente na execução deste trabalho.

Resumo

Com o passar do tempo, nota-se um aumento gradativo da demanda por veículos mais econômicos e que disponham de itens capazes de aumentar o conforto e a segurança. Cita-se, como exemplo, o controle de cruzeiro (*Cruise Control*) que, atualmente presente em diversos veículos, é responsável por controlar a velocidade do veículo de maneira autônoma, sem a necessidade de intervenção do condutor no pedal de aceleração, resultando em um aumento de conforto ao reduzir o esforço para dirigir, além de prover efetividade para manter a velocidade do veículo em torno de um valor desejado. Neste contexto, o presente trabalho apresenta o desenvolvimento de um controlador de cruzeiro para operar em um veículo modelo Volkswagen Polo Sedan 2.0L 2004, o qual não possui este recurso em seu estado de fábrica. Para a implementação deste recurso, o trabalho faz uso de uma unidade de gerenciamento eletrônico (conhecida também por *Electronic Control Unit* - ECU) desenvolvida em 2013, no âmbito do projeto Otto II (PEREIRA, 2013), para controle do motor presente no respectivo veículo, viabilizando, desta forma, a validação do controle de cruzeiro por meio de testes utilizando o veículo em um dinamômetro inercial. Entretanto, previamente ao projeto do controlador de cruzeiro, o presente trabalho teve como foco o aperfeiçoamento do funcionamento desta ECU, visando a sua evolução no que diz respeito ao estado da arte de unidades de gerenciamento eletrônico de motores. Para isto, em sua primeira fase, o trabalho realizou diversas melhorias nas malhas de controle já existentes no *firmware* de 2013, tais como controle de marcha lenta, controle da borboleta eletrônica e controle de partida. Ao mesmo tempo, novos recursos foram implementados em *firmware*: controle de torque, controle da mistura ar/combustível em malha fechada (sonda lambda), segurança na comunicação entre blocos da ECU, identificação de marcha, suporte para diagnóstico via OBD-II, dentre outros. Além destas atividades envolvendo desenvolvimento de *firmware*, o trabalho, ainda em sua primeira fase, promoveu o desenvolvimento de uma nova ferramenta de *software* que, além de ser capaz de monitorar diversos parâmetros da ECU em tempo real, integra diversas funções, tais como função de computador de bordo alternativo, opção para controle do motor através da simulação do pedal de aceleração, opção para alteração da rotação de marcha lenta e função para automatização do ensaio de identificação do veículo (tarefa necessária para o projeto do controlador de cruzeiro).

Palavras-chaves: Eletrônica automotiva. Unidade de gerenciamento eletrônico. Motor de combustão interna. Controle de cruzeiro. Controle de marcha lenta. Identificação de sistemas.

Abstract

Over time, there is a gradual increase of the demand for economical vehicles equipped with items capable of increasing the comfort and safety. As an example, the Cruise Control, which is already available in several vehicles, is responsible to control the vehicle speed in an autonomous manner, without the driver intervention on the throttle pedal. As a result, a greater comfort is achieved by reducing the effort to drive, besides providing effectiveness to keep the vehicle speed around a desired value. In this context, this project aims the development of a Cruise Control applied to a vehicle Volkswagen Polo Sedan 2.0L 2004, in which such resource is not available. To implement this resource, the project uses an electronic engine management unit (also known as Electronic Control Unit - ECU) developed in 2013 by the Otto II project ([PEREIRA, 2013](#)). This ECU is responsible to control the engine of the respective vehicle, which allows the Cruise Control validation through a set of tests performed with the vehicle on an inertial dynamometer. However, prior to the Cruise Control design, this project focused on the ECU operation enhancement, in order to achieve the state of the art in electronic engine management units. For this goal, the project, during its first phase, performed several improvements on the control algorithms already existing in the firmware developed in 2013, such as idle speed control, electronic throttle valve control and engine starting control. At the same time, new features were fully implemented in firmware: torque control, closed loop air/fuel ratio control (lambda control), safety for the communication among ECU blocks, gear identification, support to OBD-II diagnostic, among others. In addition to the firmware development activities, the project, still in its first phase, developed a new software tool capable of monitoring several ECU parameters in real time, besides providing many functions, such as alternative board computer, an option to control the engine by simulating the throttle pedal, an option to change the idle speed and a function to automate the system identification test (task required for the Cruise Control design).

Keywords: Automotive electronic. Electronic engine management unit. Internal combustion engine. Cruise Control. Idle speed control. System identification.

Lista de ilustrações

Figura 1 – Plataforma de trabalho para aprimoramento do <i>firmware</i> da ECU	25
Figura 2 – Motor Volkswagen 2.0L	26
Figura 3 – Montagem para permitir conexão direta da ECU ao chicote do motor .	27
Figura 4 – Intersecção utilizada no projeto de 2013 para conexão da ECU aos sinais do motor	28
Figura 5 – Unidade de gerenciamento eletrônico da Bosch	30
Figura 6 – Diagrama do controle de torque em uma ECU convencional	31
Figura 7 – Diagrama do controle de torque em uma ECU tipo <i>torque-based</i>	32
Figura 8 – Emissão de poluentes x fator lambda	33
Figura 9 – Curva característica da sonda lambda	34
Figura 10 – Diagrama de um controle de mistura a/c	35
Figura 11 – Figura ilustrativa do processo de auto-ignição	36
Figura 12 – Esquemático de um controle de rotação de marcha lenta	38
Figura 13 – Atuador de marcha lenta	38
Figura 14 – Válvula borboleta eletrônica	39
Figura 15 – Esquemático de um controle de posição da válvula borboleta eletrônica	40
Figura 16 – Detecção de pedestres é um exemplo de tarefa de uma tecnologia ADAS	41
Figura 17 – Exemplo de luz de acionamento do controle de cruzeiro no painel do veículo	42
Figura 18 – Botões para gerenciamento do <i>Cruise Control</i>	43
Figura 19 – Exemplo de luz de acionamento do controle de cruzeiro adaptativo no painel do veículo	44
Figura 20 – Diagrama de um controle de cruzeiro adaptativo	45
Figura 21 – Ilustração do funcionamento do controle de cruzeiro adaptativo	46
Figura 22 – O sistema <i>Audi Pre Sense Plus</i>	47
Figura 23 – Ilustração do sistema <i>Lane Departure Warning System</i>	48
Figura 24 – Motor de vibração utilizado pelo sistema <i>Audi Active Lane Assist</i> para alertar o motorista	49
Figura 25 – Sistema <i>City Park Full Automatic</i> do grupo PSA Peugeot Citroën, previsto para 2017	51
Figura 26 – Exemplo de aviso emitido pelo sistema <i>Driver Attention Alert</i> da Nissan	52
Figura 27 – Diagrama de um barramento CAN	53
Figura 28 – Adaptador OBD-II com comunicação <i>bluetooth</i>	54
Figura 29 – Aplicativo <i>Torque</i> para uso com adaptador OBD-II	55
Figura 30 – Conector OBD-II de um Ford Fiesta Sedan 2009 1.6 Flex	56
Figura 31 – Diagrama de blocos do <i>hardware</i> da ECU v2.0	58

Figura 32 – Diagrama de blocos da família de microcontroladores S12XE	60
Figura 33 – Protótipo da ECU v2.0	61
Figura 34 – Diagrama do controle de rotação do trabalho de 2013 (Projeto Otto II)	62
Figura 35 – Sinal de ignição com ilustração dos parâmetros de acionamento	64
Figura 36 – Abertura da borboleta eletrônica em função da referência de torque relativo	67
Figura 37 – Diagrama de blocos da primeira versão do controle da rotação de marcha lenta (sem atuação na ignição)	69
Figura 38 – Oscilações na rotação de marcha lenta quando se utiliza controlador PI de ganho K_i fixo para controle da borboleta eletrônica	70
Figura 39 – Fluxograma para determinação do ganho K_i do controle de posição da borboleta	71
Figura 40 – Rotação de marcha lenta quando se utiliza controlador PI de ganho K_i variável para controle da borboleta eletrônica	72
Figura 41 – Diagrama de blocos da versão final do controle da rotação de marcha lenta	74
Figura 42 – Fluxograma do controlador PI da marcha lenta (atuação na válvula borboleta eletrônica)	75
Figura 43 – Fluxograma do controlador PD da marcha lenta (atuação no avanço de ignição)	76
Figura 44 – Referência de rotação da marcha lenta em função da temperatura do motor	77
Figura 45 – Posição base da borboleta eletrônica para controle da rotação de marcha lenta	78
Figura 46 – Oscilações na posição da borboleta eletrônica quando a sua referência reduz bruscamente (controlador PI)	79
Figura 47 – Ausência de oscilações na posição da borboleta eletrônica quando a sua referência reduz bruscamente (controlador PID)	80
Figura 48 – Diagrama de blocos do controle de posição da borboleta eletrônica	81
Figura 49 – Diagrama de blocos do controle da mistura a/c em malha fechada	83
Figura 50 – Resposta da sonda lambda de banda estreita	84
Figura 51 – Fluxograma do controle lambda	86
Figura 52 – Fluxograma do controle de aquecimento da sonda lambda	88
Figura 53 – Ilustração do uso de CRC para detecção de erros na transmissão de dados entre os blocos Gerenciamento e Sincronismo	90
Figura 54 – Aplicação de monitoramento ECU v2.0 Monitor	92
Figura 55 – Seleção do período de amostragem na aplicação de monitoramento	93
Figura 56 – Parâmetros de computador de bordo calculados pela aplicação de monitoramento	95

Figura 57 – Interface gráfica da aplicação de monitoramento de 2013 (projeto Otto II)	96
Figura 58 – Interface gráfica da aplicação de monitoramento (projeto Otto IV)	96
Figura 59 – Controle do pedal de aceleração pela aplicação de monitoramento	98
Figura 60 – Seleção do modo de operação da ECU na aplicação de monitoramento	100
Figura 61 – Seleção da referência de rotação da marcha lenta na aplicação de monitoramento	101
Figura 62 – Leitura das versões de <i>firmware</i> na aplicação de monitoramento	102
Figura 63 – Registro em arquivo dos parâmetros lidos pela aplicação de monitoramento	103
Figura 64 – Arquivo de <i>log</i> da aplicação de monitoramento	104
Figura 65 – Arquivo de <i>log</i> importado na ferramenta Excel	105
Figura 66 – Controle do ensaio de identificação pela aplicação de monitoramento	106
Figura 67 – Fluxograma do controle do ensaio de identificação	108
Figura 68 – Ensaio de identificação em andamento na aplicação de monitoramento	109
Figura 69 – Visualização gráfica dos dados obtidos no ensaio de identificação	110
Figura 70 – Arquivo de <i>log</i> do ensaio de identificação importado na ferramenta Excel	111
Figura 71 – Posição da borboleta para a partida do motor	113
Figura 72 – Atraso do controle de partida	113
Figura 73 – Ganho de injeção para partida a frio	114
Figura 74 – Ganho de injeção para motor frio	115
Figura 75 – Fator de correção dos tempos de injeção e ignição em função da tensão de bateria	116
Figura 76 – Relação rotação/velocidade da 3ª marcha	118
Figura 77 – Ilustração do processo de <i>debounce</i> para filtragem de sinais digitais	119
Figura 78 – Elevação da rotação de marcha lenta para 1000 RPM quando a embreagem é acionada	121
Figura 79 – Adaptador OBD-II conectado à ECU v2.0	124
Figura 80 – Leitura de parâmetros da ECU v2.0 por meio do aplicativo <i>Torque</i>	124
Figura 81 – Botão de emergência conectado à ECU v2.0	127
Figura 82 – Curvas de potência e torque (ECU original x ECU v2.0)	129
Figura 83 – Curvas de potência e torque (modo normal x modo econômico)	130
Figura 84 – Controle da rotação de marcha lenta a 800 RPM (versão sem atuação na ignição)	131
Figura 85 – Controle da rotação de marcha lenta a 700 RPM (versão sem atuação na ignição)	132
Figura 86 – Controle da rotação de marcha lenta durante acionamento do ar condicionado (versão sem atuação na ignição)	133
Figura 87 – Controle da rotação de marcha lenta durante acionamento da carga elétrica do veículo (versão sem atuação na ignição)	134

Figura 88 – Controle da rotação de marcha lenta com variação lenta na referência (versão sem atuação na ignição)	135
Figura 89 – Controle da rotação de marcha lenta com variação rápida na referência (versão sem atuação na ignição)	136
Figura 90 – Controle da rotação de marcha lenta a 800 RPM	137
Figura 91 – Controle da rotação de marcha lenta a 700 RPM	138
Figura 92 – Controle da rotação de marcha lenta durante acionamento do ar condi- cionado	139
Figura 93 – Controle da rotação de marcha lenta durante acionamento da carga elétrica do veículo	140
Figura 94 – Controle da rotação de marcha lenta com variação lenta na referência .	141
Figura 95 – Controle da rotação de marcha lenta com variação rápida na referência	142
Figura 96 – Controle da rotação de marcha lenta a 600 RPM	143
Figura 97 – Controle de posição da válvula borboleta eletrônica com variação lenta na referência	144
Figura 98 – Controle de posição da válvula borboleta eletrônica com variação rápida na referência	145
Figura 99 – Controle de posição da válvula borboleta eletrônica com variação lenta na referência (posição abaixo de <i>Limp Home</i>)	146
Figura 100 – Controle de posição da válvula borboleta eletrônica com variação rápida na referência (posição abaixo de <i>Limp Home</i>)	147
Figura 101 – Controle lambda durante o regime de marcha lenta	148
Figura 102 – Controle lambda em torno de 1700 RPM	149
Figura 103 – Atuação do controle de partida com motor frio	150
Figura 104 – Atuação do controle de partida com motor quente	151
Figura 105 – Identificação das marchas do veículo durante aceleração de 0 a 120 km/h	152
Figura 106 – Diagrama de um sistema dinâmico com saída y , entrada u , perturbação medida w e perturbação não medida v	155
Figura 107 – Diagrama do tipo caixa preta para representação da planta	161
Figura 108 – Resposta ao degrau da planta	162
Figura 109 – Diagrama de simulação do modelo descrito por $G_v(s)$	165
Figura 110 – Comparação entre as respostas da planta e do modelo estimado	166
Figura 111 – Resposta transitória ao degrau de um sistema de controle	168
Figura 112 – Diagrama de um sistema em malha fechada com compensação	170
Figura 113 – Relação entre constante de tempo e valor de regime para um sistema de primeira ordem	171
Figura 114 – Lugar das raízes do sistema compensado com $C(z)$	173
Figura 115 – Diagrama de simulação do controle de cruzeiro	175

Figura 116–Simulação do controle de cruzeiro com degrau na referência de 40 a 50 km/h	177
Figura 117–Ausência de oscilações entre os instantes de amostragem da saída (si- mulação 40-50 km/h)	178
Figura 118–Simulação do controle de cruzeiro com degrau na referência de 50 a 80 km/h	179
Figura 119–Simulação do controle de cruzeiro com degrau na referência de 50 a 120 km/h	180
Figura 120–Diagrama de simulação do controle de cruzeiro com uso do bloco <i>Matlab</i> <i>Function</i> para implementar o controlador $C(z)$	182
Figura 121–Simulação do controle de cruzeiro com degrau na referência de 40 a 50 km/h (<i>Matlab Function</i>)	184
Figura 122–Simulação do controle de cruzeiro com degrau na referência de 50 a 80 km/h (<i>Matlab Function</i>)	185
Figura 123–Ausência de oscilações entre os instantes de amostragem da saída (si- mulação 40-50 km/h com <i>Matlab Function</i>)	186
Figura 124–Simulação do controle de cruzeiro com degrau na referência de 50 a 120 km/h (<i>Matlab Function</i>)	187
Figura 125–Diagrama de blocos do controle de cruzeiro	188
Figura 126–Botão de acionamento do controle de cruzeiro da ECU v2.0	190
Figura 127–Fluxograma do controle de cruzeiro	192
Figura 128–Controle de cruzeiro acionado a 50 km/h com variação lenta na carga do dinamômetro	194
Figura 129–Controle de cruzeiro acionado a 50 km/h com variação rápida na carga do dinamômetro	195
Figura 130–Controle de cruzeiro acionado a 100 km/h com variação lenta na carga do dinamômetro	196
Figura 131–Controle de cruzeiro acionado a 100 km/h com variação rápida na carga do dinamômetro	197
Figura 132–Uso do pedal de aceleração para ultrapassagem com o controle de cruzeiro acionado a 50 km/h	198
Figura 133–Controle de cruzeiro com aplicação de degrau na referência de 40 a 50 km/h	199
Figura 134–Controle de cruzeiro com aplicação de degrau na referência de 50 a 80 km/h	200
Figura 135–Controle de cruzeiro com aplicação de degrau na referência de 50 a 120 km/h	201

Lista de tabelas

Tabela 1	–	Relação das falhas de <i>hardware</i> detectáveis pela ECU v2.0	126
Tabela 2	–	Potência e torque máximos (ECU original x ECU v2.0)	129
Tabela 3	–	Potência e torque máximos (modo normal x modo econômico)	130

Lista de abreviaturas e siglas

a/c	ar/combustível
ACC	Adaptive Cruise Control
ADAS	Advanced Driver Assistance System(s)
b/s	bit(s) por Segundo
B	byte(s)
CAN	Controller Area Network
CC	Cruise Control
CRC	Cyclic Redundancy Check
CI	Circuito Integrado
cm	centímetro(s)
cm ³	centímetro(s) cúbico(s)
CO	monóxido(s) de carbono
cv	cavalo(s), referente à unidade de potência
ECU	Electronic Control Unit
EGO	Exhaust Gas Oxygen
EPC	Electronic Power Control, referente à luz de falha da ECU do motor presente no painel do veículo
FATEC	Faculdade de Tecnologia do Estado de São Paulo
FLEX	Flexível, referente ao sistema bicomcombustível presente nos automóveis
g	grama(s)
HC	Hidrocarboneto(s)
HMI	Human Machine Interaction
Hz	hertz
IEE	Instituto de Energia e Ambiente
ISC	Idle Speed Control
kB	quilobyte(s)
kb/s	quilobit(s) por segundo

kHz	quilohertz
km/h	quilômetro(s) por hora
km/L	quilômetro(s) por litro
kPa	quilopascal(is)
L	litro(s)
L/h	litro(s) por hora
LDWS	Lane Departure Warning System(s)
LED	Light Emitting Diode
LH	Limp Home
MAF	Mass Air Flow
MAP	Manifold Absolute Pressure
Mb/s	megabit(s) por segundo
MHz	megahertz
mkgf	metro x quilograma-força, referente à unidade de torque
ms	milissegundo(s)
NOx	Óxido(s) Nitroso(s)
OBD	On-Board Diagnostic
OMS	Organização Mundial da Saúde
ONSV	Observatório Nacional de Segurança Viária
PI	Proporcional Integral
PID	Proporcional Integral Derivativo
PWM	Pulse Width Modulation, referente à modulação por largura de pulso
RAM	Random Access Memory
RMSE	Root Mean Square Error
ROM	Read-Only Memory
RPM	rotações por minuto
RPM/s	aceleração angular
SMD	Surface Mount Device
SPI	Serial Peripheral Interface
s	segundo(s)

TPS	Throttle Position Sensor
TWC	Three Way Catalytic Converter
UART	Universal Asynchronous Receiver Transmitter
μC	microcontrolador
μs	microssegundo(s)
USB	Universal Serial Bus
USP	Universidade de São Paulo
V	volts
8v	8 válvulas, referente ao número de válvulas presente no motor
VB	Válvula Borboleta

Sumário

1	INTRODUÇÃO	20
1.1	Objetivos	21
1.1.1	Aprimoramentos na ECU	21
1.1.2	Controle de Cruzeiro	22
1.2	Justificativa do Projeto	23
1.3	Organização das Atividades	24
1.4	Plataforma de Trabalho	25
1.4.1	O motor Volkswagen 2.0L	26
1.4.2	Montagem para conexão direta da placa da ECU ao motor	27
2	REFERENCIAL TEÓRICO	29
2.1	O Controle Eletrônico do Motor a Combustão Interna	29
2.1.1	Controle de torque	30
2.1.2	Controle da mistura ar/combustível	33
2.1.3	Controle do avanço de ignição	35
2.1.4	Controle da rotação de marcha lenta	37
2.1.5	Controle de posição da válvula borboleta eletrônica	39
2.2	Sistemas Avançados de Assistência ao Motorista - ADAS	40
2.2.1	Controle de cruzeiro	41
2.2.2	Controle de cruzeiro adaptativo	43
2.2.3	Sistema pré-colisão	46
2.2.4	Sistema de aviso de mudança de faixa	48
2.2.5	Sistema de assistência a estacionamento	50
2.2.6	Sistema de detecção de sonolência	51
2.3	A Comunicação dentro do Veículo	52
2.3.1	A interface CAN	52
2.3.2	Sistema de diagnóstico de bordo - OBD	53
3	METODOLOGIA PARA APRIMORAMENTOS NA ECU	57
3.1	Descrição da ECU de 2013	58
3.1.1	Bloco de Gerenciamento	62
3.1.2	Bloco de Sincronismo	63
3.1.3	Bloco de Comunicação	65
3.2	Controle de Torque em Malha Aberta	66
3.3	Controle da Rotação de Marcha Lenta	67
3.3.1	Primeira versão	68

3.3.2	Versão final	72
3.3.3	Referência de rotação da marcha lenta	76
3.3.4	Posição inicial da borboleta eletrônica para controle da marcha lenta	77
3.4	Controle de Posição da Válvula Borboleta Eletrônica	78
3.5	Controle da Mistura Ar/Combustível (Controle Lambda)	81
3.5.1	Controle de aquecimento da sonda lambda	86
3.6	Segurança na Comunicação entre Blocos da ECU	89
3.7	Nova Aplicação de Monitoramento	91
3.7.1	Taxa de amostragem flexível	93
3.7.2	Computador de bordo alternativo	94
3.7.3	Nova interface gráfica	95
3.7.4	Controle do pedal de aceleração pela aplicação de monitoramento	97
3.7.5	Seleção do modo de operação da ECU	99
3.7.6	Seleção da referência de rotação do regime de marcha lenta	100
3.7.7	Leitura das versões de <i>firmware</i> dos blocos da ECU	101
3.7.8	Registro de parâmetros em arquivo (<i>Log</i>)	102
3.7.9	Automatização do ensaio de identificação	106
3.8	Outros Aperfeiçoamentos	112
3.8.1	Controle de partida	112
3.8.2	Estratégia para motor frio	114
3.8.3	Fator de correção de acordo com a tensão da bateria	115
3.8.4	Cálculo da velocidade do veículo	116
3.8.5	Identificação de marcha	117
3.8.6	<i>Debounce</i> de sensores digitais	118
3.8.7	Filtro para aceleração e desaceleração com veículo engatado	120
3.8.8	Elevação da marcha lenta para arranque do veículo	120
3.8.9	Aumento da troca de dados no barramento CAN	122
3.8.10	Suporte para diagnóstico via OBD-II	123
3.8.11	Deteção de falhas de hardware	125
3.8.12	Botão de emergência	127
3.9	Resultados	128
3.9.1	Controle de torque em malha aberta - Dirigibilidade	128
3.9.2	Curvas de Potência	128
3.9.3	Controle da rotação de marcha lenta	130
3.9.3.1	Primeira versão	131
3.9.3.2	Versão final	136
3.9.4	Controle de posição da válvula borboleta eletrônica	144
3.9.5	Controle da mistura ar/combustível em malha fechada (controle lambda)	148
3.9.6	Controle de partida	149

3.9.7	Identificação de marcha	151
3.10	Considerações Finais	152
4	METODOLOGIA APLICADA AO PROJETO DO CONTROLE DE CRUZEIRO	154
4.1	Identificação do Sistema	154
4.1.1	Sistemas dinâmicos	155
4.1.2	Processo de modelagem	156
4.1.2.1	Definição de entrada e saída do modelo	157
4.1.2.2	Sinal de excitação da entrada	158
4.1.2.3	Período de amostragem	159
4.1.2.4	Duração do experimento	160
4.1.3	Ensaio de identificação automatizado	160
4.1.4	Função de transferência do modelo	163
4.1.5	Validação do modelo	164
4.2	Projeto do Controlador de Cruzeiro	166
4.2.1	Definição das especificações	167
4.2.2	Projeto do controlador	169
4.2.3	Validação do controlador por meio de simulação	174
4.2.4	Simulação com o controlador implementado em <i>Matlab Function</i>	180
4.2.5	Implementação do controlador em <i>firmware</i>	187
4.2.5.1	Características do acionamento do controle de cruzeiro	189
4.2.5.2	Retomada da aceleração do veículo para ultrapassagem	190
4.2.5.3	Fluxograma da implementação em <i>firmware</i> do controle de cruzeiro	191
4.3	Resultados	193
4.3.1	Controle de cruzeiro acionado a 50 km/h	193
4.3.2	Controle de cruzeiro acionado a 100 km/h	195
4.3.3	Uso do pedal de aceleração com o controle de cruzeiro acionado	198
4.3.4	Aplicação de degrau na referência do controle de cruzeiro	199
4.4	Considerações finais	202
5	CONCLUSÕES	203
5.1	Sugestões para projetos futuros	205
	REFERÊNCIAS	206
	APÊNDICES	211
	APÊNDICE A – FIRMWARE	212

A.1	Bloco de Gerenciamento	212
A.1.1	Controle de Cruzeiro	212
A.1.2	Gerenciamento entre controle de torque e controle de marcha lenta	216
A.1.3	Controle de torque em malha aberta	218
A.1.4	Controle de marcha lenta por atuação na borboleta eletrônica	219
A.1.5	Controle de marcha lenta por atuação no avanço de ignição	222
A.1.6	Cálculo da referência de rotação da marcha lenta	224
A.1.7	Controle de posição da válvula borboleta eletrônica	227
A.1.8	Cálculo do tempo de injeção	230
A.1.9	Controle da mistura a/c em malha fechada (controle lambda)	233
A.1.10	Controle de aquecimento da sonda lambda	236
A.1.11	Controle de partida	238
A.1.12	Cálculo da velocidade do veículo	241
A.1.13	Algoritmo para identificação de marcha	242
A.1.14	Algoritmo para <i>debounce</i> de sensores digitais	243
A.1.15	Filtro da referência de posição da borboleta eletrônica	247
A.2	Bloco de Sincronismo	249
A.2.1	Atualização dos parâmetros de injeção e ignição	249
A.2.2	Interrupção do <i>timer</i> que controla os sinais de injeção 1 e 4	251
A.3	Bloco de Comunicação	256
A.3.1	Interrupção da recepção de dados da UART (processamento no XGATE)	256
A.3.2	Interrupção da recepção de dados da CAN	259
	APÊNDICE B – SOFTWARE	264
B.1	Rotina para ativação e desativação do monitoramento da ECU	264
B.2	Interrupção do <i>timer</i> de controle do monitoramento da ECU	266
B.3	Interrupção do <i>timer</i> de controle do ensaio de identificação	268

1 Introdução

Com a expansão da produção de carros movidos a motores bicom bustíveis e a preocupação cada vez maior com a emissão de gases poluentes, o Brasil vive uma demanda crescente por conhecimentos na área de eletrônica automotiva, especialmente em busca de soluções inovadoras no gerenciamento eletrônico de motores, a fim de reduzir a emissão de poluentes e melhorar a eficiência do motor, principalmente quando se utiliza o etanol como combustível. Somam-se ainda a estes fatos, exigências de usuários por veículos com maior conforto e, sobretudo, com maior segurança.

Segundo a Organização Mundial da Saúde (OMS), no Brasil foram registradas mais de 41 mil mortes no trânsito em 2013, colocando o país em primeiro lugar na América do Sul em número de mortes de trânsito por habitante ([ORGANIZAÇÃO MUNDIAL DA SAÚDE, 2015](#)). Fatores como sonolência e cansaço, por exemplo, são responsáveis por metade dos acidentes em rodovias brasileiras, de acordo com dados do Observatório Nacional de Segurança Viária (ONSV) ([OBSERVATÓRIO NACIONAL DE SEGURANÇA VIÁRIA, 2015](#)). Estes fatos indicam que é interessante explorar novas tecnologias para melhorar a segurança dos veículos e de seus tripulantes.

Neste cenário ganham importância os chamados sistemas avançados de assistência ao motorista (*Advanced Driver Assistance Systems* - ADAS) que, na última década, têm sido cada vez mais empregados em veículos novos pela indústria automobilística. Estes sistemas ganharam larga difusão graças ao avanço da tecnologia de sensores e ao aumento da capacidade computacional da eletrônica embarcada ([KILIÇ et al., 2015](#)). De modo geral, ADAS é composto por sistemas embarcados de tempo real capazes de melhorar a dirigibilidade, conforto e a segurança, ajudando a evitar, por exemplo, colisões e acidentes.

Atualmente, existem vários tipos de sistemas ADAS, cada um fornecendo diferentes recursos ao condutor. Enquanto alguns melhoram o conforto e podem evitar pequenos acidentes de trânsito, outros são mais críticos no que diz respeito à segurança do motorista. Controle de cruzeiro, controle de cruzeiro adaptativo, sistemas pré-colisão e sistemas de assistência a estacionamento são alguns exemplos de ADAS. ([SHAOUT; COLELLA; AWAD, 2011](#)).

O controle de cruzeiro (*Cruise Control*), por exemplo, corresponde a um sistema capaz de controlar a velocidade do veículo de maneira autônoma, sem a necessidade de intervenção do condutor no pedal de aceleração, resultando em um aumento do conforto do motorista ao reduzir o seu esforço para dirigir, além de prover efetividade para manter a velocidade do veículo em torno de um valor desejado ([VOLKSWAGEN, 2016b](#)).

1.1 Objetivos

O Projeto Otto IV tem como objetivo principal o desenvolvimento de um controlador de cruzeiro para operar em um veículo modelo Volkswagen Polo Sedan 2.0L 2004, o qual não possui este recurso em seu estado de fábrica. Para a implementação deste controlador, o trabalho faz uso de uma unidade de gerenciamento eletrônico (conhecida também por *Electronic Control Unit* - ECU) desenvolvida em 2013, no âmbito do projeto Otto II (PEREIRA, 2013), para controle do motor presente no respectivo veículo. Entretanto, previamente ao projeto do controle de cruzeiro, o presente trabalho tem como foco aprimorar o funcionamento desta unidade de gerenciamento eletrônico, visando, dentre outros, a sua preparação para receber a implementação do controlador de cruzeiro. Por esta razão, o trabalho, em termos de organização, se divide em duas fases, sendo a primeira responsável pelo aprimoramento da ECU de 2013, enquanto a segunda tem como objetivo o desenvolvimento do controle de cruzeiro. As seções a seguir abordam um resumo das principais tarefas a serem realizadas em cada fase do projeto.

1.1.1 Aprimoramentos na ECU

As tarefas executadas na primeira fase do projeto buscam aperfeiçoar o funcionamento da ECU desenvolvida em 2013, visando a sua evolução no que diz respeito ao estado da arte de unidades de gerenciamento eletrônico de motores. Para isto, parte das atividades desenvolvidas nesta fase são destinadas a melhorar diversas malhas de controle já existentes no *firmware* desenvolvido em 2013, tais como controle de marcha lenta, controle de posição da borboleta eletrônica e controle de partida. Outras atividades, no entanto, se concentram na implementação de novos recursos, tais como controle de torque, controle da mistura ar/combustível em malha fechada, segurança na comunicação entre blocos da ECU, compensação de parâmetros de injeção e ignição, identificação da marcha engatada, suporte para diagnóstico do sistema via protocolo OBD-II, dentre outros.

Cita-se, a título de exemplo, a substituição do controle de rotação modelado no trabalho de 2013 por um controle de torque em malha aberta, ficando o primeiro restrito apenas ao regime de marcha lenta. Espera-se, por meio desta reformulação da ECU, uma melhora significativa na dirigibilidade do veículo, promovida pelo fato da entrada do sistema (pedal de aceleração) passar a ter uma relação de proporcionalidade maior com a aceleração do veículo, característica favorável também à implementação do controle de cruzeiro.

Além das atividades envolvendo desenvolvimento de *firmware*, o trabalho, em sua primeira fase, promove o desenvolvimento de uma nova aplicação (*software*) para monitoramento em tempo real da ECU. A seguir foram destacados os principais recursos disponibilizados ao desenvolvedor por meio desta aplicação:

- Capacidade para monitorar, em tempo real, diversos parâmetros do sistema, incluindo aqueles que não pertencem à ECU do motor (setas, luz de bateria, farol, freio de estacionamento, portas do veículo, hodômetro, nível de combustível, dentre outros);
- Cálculo de parâmetros de computador de bordo como, por exemplo, consumo médio e instantâneo do motor;
- Função para controle do motor através da simulação do pedal de aceleração;
- Função para alterar a rotação do regime de marcha lenta, mesmo enquanto o motor estiver operando;
- Função para alterar o comportamento da ECU, tornando-a mais econômica ou mais esportiva em termos de dirigibilidade;
- Função para automatização do ensaio de identificação do veículo, tarefa necessária para o projeto do controlador de cruzeiro.

Além de favorecer a implementação do controle de cruzeiro, espera-se que os resultados obtidos nesta primeira fase do trabalho sejam decisivos para melhorar consideravelmente o funcionamento do motor, de tal modo que o seu comportamento seja bem próximo daquele obtido quando se utiliza a sua ECU original de fábrica. Com isto, o projeto atual assume o compromisso de disponibilizar uma ECU didática para uso em futuros projetos na área de eletrônica automotiva, de tal forma que estes tenham à disposição uma ECU robusta e estável para ser utilizada como plataforma de trabalho.

Reitera-se ainda a importância do trabalho na formação de mão de obra com capacitação técnica para fomentar a indústria automobilista brasileira, sobretudo na área de eletrônica automotiva, visto que a execução do projeto envolve o conhecimento de diversas áreas da engenharia, além de exigir também o conhecimento e o desenvolvimento prático na área de gerenciamento eletrônico de motores.

1.1.2 Controle de Cruzeiro

Conforme mencionado, a segunda fase do projeto tem como objetivo o desenvolvimento de um controlador de cruzeiro para operação em um automóvel Polo Sedan 2.0L 2004, o qual não conta com este recurso na sua versão de fábrica. Para isto, a tarefa inicial desta fase consiste na obtenção de um modelo que represente satisfatoriamente o sistema constituído pelo veículo equipado com a ECU deste trabalho, viabilizando, desta forma, o posterior projeto de controle. Neste contexto, o trabalho se baseia em um processo de modelagem empírico denominado identificação de sistemas, o qual envolve o uso do próprio veículo em um experimento cujo controle é de responsabilidade da aplicação de monitoramento desenvolvida na primeira fase do trabalho.

De posse de um modelo para o sistema em questão, a segunda fase do trabalho prossegue com o projeto do controlador na ferramenta MATLAB, sendo a sua validação inicial realizada por meio de simulações, as quais têm como finalidade verificar se as especificações definidas para a aplicação de controle estão sendo devidamente atendidas. Uma vez validado em ambiente de simulação, o controlador é implementado no *firmware* da ECU aprimorada na primeira fase do trabalho, viabilizando, desta forma, a validação do controle de cruzeiro por meio de testes utilizando o veículo em conjunto com um dinamômetro inercial.

Como resultado final, espera-se que o controle projetado seja capaz de manter a velocidade do veículo a um determinado valor de referência, inclusive em situações de carga. Além disto, o controle de cruzeiro deve permitir que o motorista assuma o controle da aceleração quando assim se fizer necessário (situação de ultrapassagem, por exemplo), devendo o sistema retomar automaticamente o controle da velocidade quando o pedal de aceleração for liberado.

1.2 Justificativa do Projeto

Atualmente, a demanda por conhecimentos na área de eletrônica automotiva no Brasil está cada vez maior. Projetos inovadores que necessitariam intervir diretamente no controle do motor se tornam muito complicados de serem implementados em unidades de gerenciamento presentes hoje no mercado, dado que estas são tratadas com sigilo intelectual absoluto por parte de fabricantes de veículos. Sendo assim, se torna muito difícil o trabalho de pesquisa visando inovação na área, o que expõe uma grande deficiência do país em conhecimentos específicos na área de eletrônica automotiva. Grande parte do desenvolvimento acadêmico no setor é fundamentado em suposições teóricas e engenharia reversa.

Em 2012, a Escola Politécnica da USP iniciou, através do projeto Otto (SCARPINETTI; SOARES, 2012), o desenvolvimento de unidades de gerenciamento eletrônico de motores a combustão interna, dedicando-se inicialmente ao desenvolvimento de uma unidade para operar em um motor GM 1.8 montado em bancada¹. Como resultado, o projeto Otto entregou um módulo eletrônico capaz de manter o respectivo motor funcionando corretamente em regime de marcha lenta, além de possibilitar o controle da sua rotação até 5000 RPM (SCARPINETTI; SOARES, 2012).

No ano de 2013, o projeto Otto II (PEREIRA, 2013) propôs o desenvolvimento de uma nova unidade de gerenciamento eletrônico para controle de um motor 2.0L a gasolina de um veículo Volkswagen Polo Sedan 2004, visando aprimorar a estratégia de controle do motor em situações de operação com carga. Como resultado, a ECU desenvolvida foi

¹ Esta montagem também é conhecida por *mockup*.

capaz de controlar satisfatoriamente o respectivo motor, permitindo a condução do veículo em pista sem qualquer auxílio da sua ECU original de fábrica, além de fornecer potência e torque máximos similares aos valores nominais do motor durante a realização de testes de potência. Para alcançar estes resultados positivos, o trabalho envolveu o desenvolvimento de um *hardware* com uso de microcontroladores automotivos (PEREIRA, 2013).

Reitera-se que diversos outros trabalhos ocorreram na área de gerenciamento de motores a combustão interna como, por exemplo, o projeto proposto por Pereira (2015) em 2015 para controle da mistura ar/combustível em malha fechada. Outro exemplo corresponde ao projeto Otto III que, proposto por Santos e Rossetti (2015) também em 2015, desenvolveu um *software* de aplicação para gerenciamento de um motor Volkswagen 1.6L fazendo uso de um sistema conhecido por *FlexECU*, cujo *hardware* corresponde ao mesmo utilizado em uma ECU comercial de desenvolvimento (SANTOS; ROSSETTI, 2015).

Com intuito de dar continuidade ao desenvolvimento na área de eletrônica automotiva, o projeto visa agora aperfeiçoar a solução proposta no projeto Otto II, buscando a sua evolução no que diz respeito ao estado da arte de unidades de gerenciamento eletrônico de motores. Para isto, o *firmware* desenvolvido em 2013 para controle do motor Volkswagen 2.0L passou por uma profunda mudança em sua estrutura para receber diversas melhorias e novos recursos, sendo que o principal deles corresponde ao controle de cruzeiro. Desta forma, o trabalho se constituiu em uma oportunidade única para implementar e testar um recurso de assistência ao motorista (ADAS) que não está presente originalmente no veículo utilizado no trabalho. Vale lembrar ainda que a inclusão deste item seria uma tarefa complicada de ser realizada com a unidade original do veículo, pois neste caso não seria possível intervir diretamente no controle do motor, conforme mencionado anteriormente.

A elaboração e execução foram feitas em cooperação com o departamento de eletrônica automotiva da Faculdade de Tecnologia (FATEC) de Santo André e o Instituto de Energia e Ambiente (IEE) da Universidade de São Paulo (USP), os quais cederam, respectivamente, o veículo Polo Sedan 2004 e o dinamômetro inercial para testes.

1.3 Organização das Atividades

1. Aprimoramentos na ECU de 2013

- a) Melhorias em malhas de controle já existentes no *firmware*;
- b) Implementação de novos recursos em *firmware*;
- c) Desenvolvimento de nova ferramenta de *software*;
- d) Validação por meio de testes com o veículo em dinamômetro inercial.

2. Projeto do Controle de Cruzeiro

- a) Identificação do sistema;
- b) Definição das especificações da aplicação de controle;
- c) Projeto do controlador na ferramenta MATLAB;
- d) Validação do controlador na ferramenta Simulink;
- e) Implementação do controlador em *firmware*;
- f) Validação por meio de testes com o veículo em dinamômetro inercial.

1.4 Plataforma de Trabalho

Para realização das atividades listadas na [seção 1.3](#), o projeto faz uso da plataforma de trabalho destacada na [Figura 1](#). Constituída de um veículo modelo Volkswagen Polo Sedan 2.0L 2004 aplicado em um dinamômetro inercial, esta plataforma permite aprimorar a estratégia de *firmware* da ECU, avaliar o desempenho do motor em condições de carga e realizar diversos testes envolvendo o controle de cruzeiro, cujo desenvolvimento será abordado no [Capítulo 4](#).

Figura 1 – Plataforma de trabalho para aprimoramento do *firmware* da ECU.

Fonte: Autor.

1.4.1 O motor Volkswagen 2.0L

O veículo utilizado para testes é equipado com um motor Volkswagen 2.0L 8v, exibido na [Figura 2](#). Dentre as diversas características deste motor, destacam-se:

- Operação com gasolina;
- Cilindrada de 1984 cm^3 ([PORTAL VRUM, 2013](#));
- 116 cv de potência máxima a 5200 RPM ([PORTAL VRUM, 2013](#));
- Torque máximo de 17,3 mkgf a 2400 RPM ([PORTAL VRUM, 2013](#));
- 4 cilindros em linha ([PORTAL VRUM, 2013](#));
- 2 válvulas por cilindro, característico do motor 8 válvulas (8v) ([PORTAL VRUM, 2013](#));
- 2 bobinas de ignição dupla, com acionamento banco-a-banco (acionamento simultâneo nos cilindros 1-4 e 2-3, com “centelha perdida” em um dos cilindros);
- Injeção multiponto sequencial 1-3-4-2 ([PORTAL VRUM, 2013](#)).

Figura 2 – Motor Volkswagen 2.0L.

Fonte: Autor.

1.4.2 Montagem para conexão direta da placa da ECU ao motor

A montagem da [Figura 3](#) foi construída ao decorrer deste trabalho para facilitar a conexão da placa da ECU ao chicote do motor, além de reduzir problemas com ruídos e mal contatos de sinais elétricos. Como resultado, o tempo de conexão da placa aos sinais do motor foi drasticamente reduzido, em comparação ao caso em que se utiliza a intersecção da [Figura 4](#) para conexão destes sinais, procedimento este que foi muito utilizado em trabalhos anteriores. O lado direito da [Figura 2](#) destaca a ECU deste trabalho conectada diretamente ao motor por meio da montagem da [Figura 3](#).

Os créditos da referida montagem são destinados a Bruno Silva Pereira, autor do trabalho de 2015 ([PEREIRA, 2015](#)), e Demerson Moscardini.

Figura 3 – Montagem para permitir conexão direta da ECU ao chicote do motor.

Fonte: Autor.

Figura 4 – Intersecção utilizada no projeto de 2013 para conexão da ECU aos sinais do motor.

Fonte: Autor (2013).

2 Referencial Teórico

Nota-se, com o passar do tempo, um aumento gradativo dos componentes elétricos e eletrônicos nos automóveis, que se deve basicamente ao aumento da complexidade e do poder de processamento destes elementos. Estima-se que, no futuro, mais de 30% dos componentes de um veículo sejam de natureza eletrônica, contribuindo de modo geral para a redução do consumo de combustível e emissão de poluentes, além de melhorar determinados fatores como conforto, dirigibilidade e segurança dos passageiros (GUZZELLA; ONDER, 2004).

2.1 O Controle Eletrônico do Motor a Combustão Interna

O uso de componentes eletrônicos para o controle do motor a combustão interna se mostrou essencial para o alcance da máxima potência com o menor consumo de combustível e menor emissão de poluentes. Entretanto, ressalta-se que, mesmo com o gerenciamento eletrônico, o motor a combustão interna não apresenta rendimentos elevados devido, principalmente, às elevadas perdas de energia que ocorrem, em sua maior parte, na forma de calor (MANAVELLA, 1996).

De modo geral, a unidade de gerenciamento eletrônico do motor, ilustrada na Figura 5 e conhecida também por *Electronic Control Unit* (ECU), é responsável por ajustar todas as funções do motor, de modo a garantir o menor consumo de combustível e o menor nível de emissão de poluentes, ao mesmo tempo que entrega ao condutor um bom desempenho em termos de dirigibilidade. Uma vez em operação, a ECU monitora em tempo real todos os parâmetros do sistema que influenciam na segurança, no desempenho e nas emissões de poluentes, ajustando-os da melhor maneira possível para otimizar a dirigibilidade do veículo (ROBERT BOSCH GMBH, 2017).

Um exemplo de melhoria em termos de dirigibilidade consiste no fato da ECU controlar o motor para que o torque fornecido seja proporcional à posição do pedal de aceleração, de maneira a manter uma relação de proporcionalidade também à aceleração do veículo. Outro exemplo consiste na redução de solavancos no veículo como resultado da atuação da ECU no sentido de suavizar acelerações e desacelerações acentuadas.

Figura 5 – Unidade de gerenciamento eletrônico da Bosch.

Fonte: [Robert Bosch GmbH \(2017\)](#).

Integrando injeção eletrônica e ignição eletrônica em um único módulo, as primeiras ECUs surgiram com funções muito básicas. Entretanto, com as exigências de leis locais para a redução da emissão de poluentes e redução do consumo de combustível, aliado à demanda de novos veículos com maior potência e maior segurança, novas funções foram adicionadas às ECUs existentes. Como exemplo, cita-se o sistema de controle *Motronic* da Bosch que, ao longo do tempo, recebeu novas funções como regulagem da rotação de marcha lenta, controle de estequiometria, controle do eixo de comando para redução da emissão de gases de escape, dentre outros ([BOSCH, 2005](#)).

A seguir se destacam alguns dos principais controles presentes em uma unidade de gerenciamento eletrônico do motor.

2.1.1 Controle de torque

Segundo [Guzzella e Onder \(2004\)](#), a tarefa mais importante de um motor é fornecer torque, de modo a atender a um certo valor de demanda que, em geral, é definida pelo condutor através do pedal de aceleração do veículo. Entretanto, diversos outros subsistemas envolvidos no controle do motor (como, por exemplo, o controle de cruzeiro)

podem influenciar nesta demanda de torque. Assim, surge o conceito de ECU baseada em torque (*torque-based ECU*), que corresponde a uma evolução da ECU convencional.

A Figura 6 mostra a estrutura do controle de torque utilizada em ECUs convencionais. Nota-se que existem diversos laços de controles sendo executados em paralelo, além do fato de a estrutura possuir diversos fluxos de sinais, o que evidencia a complexidade destes sistemas. Além disto, ressalta-se a dificuldade na parametrização da estrutura de *software* deste tipo de ECU, o que expõe um grande problema na sua utilização em novos motores¹ (GUZZELLA; ONDER, 2004).

Figura 6 – Diagrama do controle de torque em uma ECU convencional.

Fonte: Guzzella e Onder (2004).

¹ Modificar uma ECU convencional para atender um novo motor demandaria um grande esforço, visto que exigiria modificações em praticamente todos os blocos que a integram.

A Figura 7 exibe a estrutura do controle de torque utilizada em ECUs *torque-based*, que, diferentemente da estrutura de ECUs convencionais, se mostra mais simples em termos de fluxograma. Neste sistema, as estratégias da ECU são desacopladas dos controles de baixo nível, responsáveis pela geração correta dos comandos dos atuadores. Assim, as malhas de controle de alto nível (marcha lenta, controle de cruzeiro, limitador de velocidade, dirigibilidade, etc.) geram um sinal de demanda de torque que é propagado até o controlador de torque através do bloco *Torque Demand Manager*. Por fim, o controlador de torque, representado pelo bloco *Torque Conversion Manager*, comanda os atuadores do motor (abertura da borboleta, injeção, ignição, etc.) de modo a garantir que o valor de demanda seja alcançado pelo motor da melhor maneira possível (GUZZELLA; ONDER, 2004).

Figura 7 – Diagrama do controle de torque em uma ECU tipo *torque-based*.

Fonte: Guzzella e Onder (2004).

Se comparada à estrutura anterior da ECU convencional (Figura 6), esta estrutura de software (Figura 7) se torna mais fácil de ser compreendida e mantida por outros desenvolvedores, uma vez que os algoritmos implementados nesta ECU refletem algo mais próximo da física dos processos inerentes ao motor (GUZZELLA; ONDER, 2004). É importante ressaltar que a implementação da ECU *torque-based* somente é possível mediante o emprego de uma válvula borboleta eletrônica, cujo funcionamento será abordado na subseção 2.1.5.

2.1.2 Controle da mistura ar/combustível

O controle da estequiometria da mistura ar/combustível (a/c) é essencial para se minimizar emissões de poluentes e proporcionar economia de combustível. Com base no gráfico da [Figura 8](#), pode-se estabelecer as seguintes considerações em relação à faixa de valor em que se encontra o fator lambda (λ):

- $\lambda < 1$: Emissão elevada de hidrocarbonetos (HC) e monóxido de carbono (CO). Também entrega alta potência devido à alta quantidade de combustível injetada;
- $\lambda = 1$: Reagentes da combustão em proporção estequiométrica, com emissões de poluentes minimizadas (principalmente com uso de um catalisador para tratar os gases de escape). Além disto, fornece uma razoável potência;
- $\lambda > 1$: Apresenta boa eficiência devido à maior massa de ar admitida pelo motor. Todavia ocorre aumento considerável da emissão de óxidos nitrosos (NO_x), o que pode comprometer a vida útil do catalisador do veículo.

Figura 8 – Emissão de poluentes x fator lambda.

Fonte: [Kiencke e Nielsen \(2005\)](#).

Em geral, a sonda lambda é o sensor responsável pelo controle em malha fechada da mistura a/c no motor. Este sensor é capaz de medir a concentração de oxigênio nos gases resultantes da combustão ([ROBERT BOSCH GMBH, 2013](#)), informando se a mistura está

rica² ou pobre³. A Figura 9 exibe a curva característica de uma sonda lambda do tipo estreita, a qual estabelece uma relação entre tensão fornecida pelo sensor e fator lambda (λ).

Figura 9 – Curva característica da sonda lambda.

Fonte: Robert Bosch GmbH (2013).

A ECU não pode, no entanto, levar em conta somente o sensor da sonda lambda para dosar a quantidade correta de combustível a ser injetada, dado que o atraso entre a injeção de combustível e a resposta da sonda lambda é consideravelmente elevado (GUZZELLA; ONDER, 2004). Por esta razão, se faz necessário uma estimativa rápida da massa de combustível a ser injetada com base, principalmente, em informações de sensores presentes no coletor de admissão do motor, responsáveis por fornecer dados para estimar a massa de ar admitida pelo motor (ASHOK; ASHOK; KUMAR, 2016).

Portanto, quando o motor estiver em regimes transitórios, o controle da mistura a/c é realizado somente em malha aberta (*feedforward control*), uma vez que a determinação da dosagem de combustível necessita ser suficientemente rápida para acompanhar as variações dos sensores durante este regime de operação. Entretanto, em caso de regime estável, o controle da mistura do motor é realizado em malha aberta e, ao mesmo tempo, em malha fechada (*feedback control*) com a informação proveniente da sonda lambda, corrigindo pre-

² Mistura rica equivale a um fator lambda (λ) menor que 1, o que indica maior quantidade de combustível em relação à quantidade de ar.

³ Mistura pobre equivale a um fator lambda (λ) maior que 1, o que indica maior quantidade de ar em relação à quantidade de combustível.

cisamente qualquer erro indesejável resultante do primeiro controle (estimação) (ASHOK; ASHOK; KUMAR, 2016).

A Figura 10 ilustra o diagrama de um controle de mistura a/c em que o combustível base (*base fuel*) estimado pelo controle *feedforward* é corrigido pelo controle *feedback*.

Figura 10 – Diagrama de um controle de mistura a/c.

Fonte: Ashok, Ashok e Kumar (2016).

2.1.3 Controle do avanço de ignição

O controle do avanço da ignição é fundamental para o correto funcionamento do motor, na medida em que influencia fortemente o seu desempenho, o consumo de combustível e o nível de emissão de poluentes (MANAVELLA, 1996).

Segundo Guzzella e Onder (2004), a eficiência do motor é limitada, dentre outros fatores, pelo fenômeno da detonação, conhecido também por *Knock*. Este fenômeno é decorrente de um processo de auto-ignição (Figura 11) que pode ocorrer internamente ao cilindro do motor. Para evitar a detonação, além de manter a taxa de compressão sempre abaixo de um valor seguro, é necessário atuar de modo *offline* (calibração) e *online* (controle em malha fechada) no avanço da ignição, parâmetro este que determina o instante em que ocorre a centelha da combustão (GUZZELLA; ONDER, 2004).

Figura 11 – Figura ilustrativa do processo de auto-ignição.

Fonte: [Carbon Cleaning USA](#) (2016).

Uma primeira calibração do avanço de ignição é realizada levando em conta a carga e a rotação do motor. Nesta etapa, o avanço de ignição é otimizado através de experimentos para minimizar a emissão de poluentes, ao mesmo tempo que se suprime qualquer fenômeno de detonação. Uma vez determinado o melhor avanço de ignição para uma dada rotação e carga, o seu valor é então armazenado em tabelas na memória da ECU. Além da rotação e da carga do motor (que pode ser estimada pela pressão do coletor), outros fatores podem influenciar na determinação do avanço de ignição, como a razão estequiométrica da mistura, temperatura do ar, tensão da bateria e temperatura do motor (KIENCKE; NIELSEN, 2005).

Uma vez finalizado o processo de calibração, o avanço da ignição passa a ser corrigido de modo *online* pela ECU durante todo o tempo em que o motor estiver em funcionamento. Esta correção é realizada através de um controle em malha fechada realimentado por um sensor de *Knock*, capaz de monitorar a combustão pela detecção do fenômeno da detonação. Assim, caso seja detectado o *Knock* (detonação), o avanço da ignição deve ser atrasado, de maneira a mantê-lo sempre a um valor seguro e, ao mesmo tempo, eficiente para o consumo de combustível e desempenho do motor (ASHOK; ASHOK; KUMAR, 2016). É importante ressaltar também que o avanço da ignição é utilizado também para o controle da rotação de marcha lenta, que será abordado na [subseção 2.1.4](#).

2.1.4 Controle da rotação de marcha lenta

O regime de marcha lenta é um dos modos de operação mais importantes do motor a combustão interna. Estima-se que, ao se dirigir no trânsito das grandes cidades, cerca de 30% do consumo total de combustível é gasto durante o regime de marcha lenta, podendo este percentual ainda aumentar com a piora do trânsito no futuro, devido ao aumento do número de veículos nas vias (ZHU et al., 2015).

O controle da rotação de marcha lenta, conhecido na literatura também por *Idle Speed Control* (ISC), deve ser robusto o suficiente para manter a rotação ao seu valor mínimo sem comprometimento da estabilidade do motor, a fim de gerar uma economia de combustível e redução de poluentes (YE, 2007). Sendo assim, o controle deve reagir rapidamente para que perturbações típicas deste regime de operação, como variação de cargas elétricas (direção, ventoinha, luzes, etc.), variação na carga do compressor de ar condicionado e alternador, não provoquem variações na rotação de marcha lenta, o que poderia elevar as vibrações do motor, provocando um desconforto ao motorista ou até mesmo o afogamento do motor (XIONG et al., 2009).

A Figura 12 ilustra o diagrama do controle de rotação do regime de marcha lenta. Nele se observa que a rotação deste regime é controlada mediante atuação no avanço do ângulo de ignição (*spark advance*) e na massa de ar admitida pelo motor, sendo esta controlada pela válvula borboleta de admissão (ASHOK; ASHOK; KUMAR, 2016). Neste último caso, o controle de marcha lenta fornece uma referência de abertura (θ_{req}) para o controle da válvula borboleta eletrônica, representado na Figura 12 pelo bloco *ETC Module*.

O efeito do avanço de ignição na rotação de marcha lenta é consideravelmente mais rápido do que o efeito da válvula de admissão, o que faz do controle da ignição um método importante para o controle da rotação de marcha lenta em regimes transitórios. Entretanto, a variação da massa de ar é capaz de variar o torque em proporções maiores se comparado à variação do ângulo da ignição. Como resultado, a correção da marcha lenta é realizada em um primeiro instante (regime transitório) pelo avanço de ignição, o qual retorna ao seu valor nominal (tabelado) conforme a borboleta eletrônica começa a atuar no sentido de corrigir o valor de regime estacionário da rotação de marcha lenta (ASHOK; ASHOK; KUMAR, 2016).

Figura 12 – Esquemático de um controle de rotação de marcha lenta.

Fonte: [Ashok, Ashok e Kumar \(2016\)](#).

Em alguns sistemas com válvula borboleta mecânica, parte da massa de ar admitida pelo motor é controlada por uma válvula de desvio (*bypass*) conhecida por atuador de marcha lenta ([Figura 13](#)), montada no corpo da borboleta ou no coletor de admissão. Assim, quando o condutor libera o pedal e a borboleta principal se fecha, a válvula *bypass* passa a ser responsável pelo controle da massa de ar admitida pelo motor durante o regime de marcha lenta ([ASHOK; ASHOK; KUMAR, 2016](#)).

Figura 13 – Atuador de marcha lenta.

Fonte: [Dias \(2012\)](#).

Em sistemas mais modernos, o controle do fluxo de ar no regime da marcha lenta é realizado diretamente por uma válvula borboleta eletrônica. Neste caso, a ECU controla

toda a massa de ar admitida pelo motor, uma vez que tem pleno controle sobre o ângulo de abertura da borboleta (ASHOK; ASHOK; KUMAR, 2016).

2.1.5 Controle de posição da válvula borboleta eletrônica

A válvula borboleta eletrônica, exibida na Figura 14, é utilizada pela ECU para controle do fluxo de ar admitido pelo motor. Nos sistemas equipados com este dispositivo, o motorista não controla diretamente o ângulo de abertura da válvula de admissão, como ocorria nos veículos com válvula de admissão mecânica. Em vez disto, o pedal envia um sinal à ECU que, por sua vez, processa e envia um sinal de controle à borboleta eletrônica, posicionando-a da melhor maneira possível na posição desejada pela estratégia de controle da ECU. Geralmente, o sinal do pedal eletrônico é compreendido pela ECU como uma referência de torque (ASHOK; ASHOK; KUMAR, 2016).

Figura 14 – Válvula borboleta eletrônica.

Fonte: Autor (2013).

Pelo fato de poder ser totalmente controlada pela ECU, esta válvula foi fundamental para a implementação de diversos recursos baseados na dinâmica do veículo (como, por exemplo, o controle de tração), além de proporcionar melhor dirigibilidade, redução do consumo de combustível e diminuição da emissão de poluentes (DEUR et al., 2004).

A posição desta válvula eletrônica é controlada através da tensão média aplicada a um servomotor que, em conjunto com uma mola interna, determina a abertura do mecanismo. Esta tensão média pode ser implementada através de um pulso de amplitude fixa com largura modulada (PWM - *Pulse-Width Modulation*), dado que este tipo de

signal pode ser facilmente gerado em microcontroladores. Além disto, a válvula já contém integrado um sensor de posição conhecido por *Throttle Position Sensor* (TPS), que corresponde a um potenciômetro responsável por fornecer realimentação para o controle em malha fechada (ASHOK; ASHOK; KUMAR, 2016), conforme exibe o diagrama da Figura 15.

O projeto do controlador deve, entre outros fatores, levar em conta características não-lineares do servomecanismo como o atrito, a mola de retorno e a folga entre as engrenagens, características estas que afetam consideravelmente o desempenho do sistema (ASHOK; ASHOK; KUMAR, 2016). Vale ressaltar que estas não linearidades se tornam mais expressivas para operações de pequenos sinais, quando se necessita que o controle responda com precisão a pequenas variações na abertura da borboleta. Cita-se, como exemplo, o controle da borboleta para o regime de marcha lenta, que exige variações na posição da válvula da ordem de $0,5^\circ$ (DEUR et al., 2004).

Figura 15 – Esquemático de um controle de posição da válvula borboleta eletrônica.

Fonte: Ashok, Ashok e Kumar (2016).

2.2 Sistemas Avançados de Assistência ao Motorista - ADAS

A cada ano, a indústria automobilística investe milhões em tecnologias de ponta que têm como finalidade aumentar a segurança dos condutores e diminuir a probabilidade de ocorrência de acidentes envolvendo seus veículos. Este conjunto de tecnologias é conhecido por sistemas avançados de assistência ao motorista (*Advanced Driver Assistance Systems - ADAS*) (SHAOUT; COLELLA; AWAD, 2011).

Formado por complexos dispositivos embarcados, os sistemas de assistência ao motorista são capazes de ler e processar entradas⁴ de diversos módulos presentes no veículo, gerando saídas em tempo real que podem variar desde um aviso luminoso até um sinal de

⁴ Em alguns casos, estas entradas podem incluir as requisições do motorista.

comando nos freios do veículo. Falhas no processamento destes sistemas podem resultar em acidentes (SHAOUT; COLELLA; AWAD, 2011).

Deteção e classificação de objetos (Figura 16), geração de alertas ao motorista em relação às condições da via e diminuição da velocidade do veículo são exemplos de tarefas executadas por tecnologias ADAS (NVIDIA, 2016).

Figura 16 – Deteção de pedestres é um exemplo de tarefa de uma tecnologia ADAS.

Fonte: NVIDIA (2016).

Como mencionado anteriormente, existem diversos tipos de sistemas ADAS, cada um provendo um recurso diferente ao condutor. Enquanto alguns melhoram o conforto e podem evitar pequenos acidentes de trânsito, outros são mais críticos no que diz respeito à segurança do motorista (SHAOUT; COLELLA; AWAD, 2011). As seções seguintes destacam alguns exemplos de sistemas de assistência ao motorista.

2.2.1 Controle de cruzeiro

O controle de cruzeiro (*Cruise Control* - CC), inventado no início do século XX com uso de componentes mecânicos, é um tipo de sistema ADAS cuja finalidade é manter, de maneira autônoma, a velocidade do veículo em um valor desejado, sem a necessidade de intervenção do motorista no pedal de aceleração. A versão eletrônica deste controle foi projetada pela primeira vez no final dos anos 60 por Kazumasa Nakamura, fazendo uso de um microprocessador de 2kB de memória ROM e 64B de memória RAM. Entretanto, somente a partir dos anos 80 se concretizou a sua produção em massa para os automóveis (SHAOUT; COLELLA; AWAD, 2011).

Figura 17 – Exemplo de luz de acionamento do controle de cruzeiro no painel do veículo.

Fonte: Volkswagen (2016d).

O projeto do controlador de cruzeiro deve ser robusto o suficiente para garantir que a velocidade do veículo esteja o mais próximo possível do seu valor de referência, incluindo situações em que estão presentes perturbações externas (vento, mudança de inclinação da pista, dentre outros). Além disto, o controle deve ser robusto para compensar variações de parâmetros do sistema, como a massa do veículo, velocidade de referência e a relação da marcha engatada (GUZZELLA; ONDER, 2004).

Dentre as vantagens do controle de cruzeiro, destacam-se:

- Aumento do conforto, pelo fato de o condutor não precisar controlar o pedal do acelerador durante todo o tempo de viagem, tornando-a menos cansativa (OSMAN; RAHMAT; AHMAD, 2009);
- Redução do esforço para dirigir, ao permitir que o controle da velocidade do veículo seja realizado através de botões, em vez de pedais (OSMAN; RAHMAT; AHMAD, 2009);
- Efetividade para automaticamente manter a velocidade do veículo dentro do limite permitido na via, mesmo em longos trajetos, evitando assim problemas com infrações de trânsito que poderiam ocorrer, de forma inconsciente, se o controle da velocidade fosse feito pelo condutor através dos pedais (VOLKSWAGEN, 2016b).

Para o gerenciamento do controle de cruzeiro, o condutor do veículo conta com botões que, geralmente, estão presentes no volante ou no painel do veículo. A Figura 18 mostra os botões de gerenciamento do *Cruise Control* de um Toyota, que podem ser utilizados para regulação da velocidade de referência do sistema, ao mesmo tempo que podem também ligar ou desligar o controle. Analogamente aos veículos Toyota, o controle de cruzeiro de veículos Volkswagen também são preparados para memorizar e manter o veículo a uma determinada velocidade, com flexibilidade para aumentar ou diminuir manualmente este valor, além de contar com uma opção para retornar o sistema ao valor de referência mais recente (VOLKSWAGEN, 2016b).

Figura 18 – Botões para gerenciamento do *Cruise Control*.

Fonte: [Toyota Motor Australia \(2016\)](#).

É importante ressaltar que, por motivos de segurança, não se aconselha o uso do controle de cruzeiro em situações de trânsito intenso, muito menos em vias com condições adversas (chuva intensa, neve, buracos na via, etc) ([VOLKSWAGEN, 2016b](#)). Caso seja necessária uma frenagem devido, por exemplo, a outro veículo mais lento à frente, o condutor poderá desarmar o controle de cruzeiro simplesmente acionando o pedal do freio, como ocorre nos veículos Toyota ([TOYOTA MOTOR AUSTRALIA, 2016](#)). Além do freio, em alguns carros manuais o desarme do controle de cruzeiro também pode ser realizado por meio do pedal da embreagem ([VOLKSWAGEN, 2016b](#)).

2.2.2 Controle de cruzeiro adaptativo

Conhecido no passado por *Intelligent Cruise Control* ou *Autonomous Intelligent Cruise Control*, o controle de cruzeiro adaptativo (*Adaptive Cruise Control – ACC*) é considerado como uma extensão do controle de cruzeiro convencional (CC). Com um número maior de sensores e malhas de controle, este sistema é capaz de detectar objetos à frente do veículo, com intuito de regular a velocidade para manter uma distância segura entre os automóveis ([VIJAY et al., 2010](#)). Entretanto, se nenhum veículo é detectado à frente, o sistema ACC funciona de maneira análoga ao controle de cruzeiro convencional ([SHAOUT; COLELLA; AWAD, 2011](#)).

Figura 19 – Exemplo de luz de acionamento do controle de cruzeiro adaptativo no painel do veículo.

Fonte: Volkswagen (2016d).

No sistema ACC, podem ser empregados, dentre outros, sensores do tipo *laser* e sensores do tipo radar. Os sensores *laser* são mais vantajosos por serem menores e mais baratos do que os radares, reduzindo o custo do sistema e a dificuldade do processo de montagem no veículo. Entretanto, apesar de mais caros e com dimensões maiores, os radares possuem maior alcance e fornecem medições mais precisas, principalmente em condições adversas como mudanças no clima e acúmulo de sujeira nas lentes (SHAOUT; COLELLA; AWAD, 2011).

Como mencionado, o sistema ACC é responsável por ajustar a sua velocidade de referência ao considerar a aproximação de outro veículo à frente. A complexidade deste sistema é maior em comparação ao sistema CC, dado que agora é necessário o processamento de um número maior de entradas, gerando controle para um número maior de atuadores, ao envolver também o comando dos freios do veículo.

A Figura 20 ilustra o diagrama de um controle de cruzeiro adaptativo, em que é possível observar atuação tanto no acelerador como nos freios, com base nas informações de velocidade e de sensores radares. Nota-se que as requisições do condutor, agrupadas no bloco denominado HMI (*Human Machine Interaction*), também são consideradas no processamento do sistema.

Figura 20 – Diagrama de um controle de cruzeiro adaptativo.

Fonte: Kılıç et al. (2015).

Cita-se, como exemplo, o sistema ACC de veículos Volkswagen, ilustrado na Figura 21. Com auxílio de um sensor radar de alcance de até 200 metros e ângulo de feixe de 12° , a unidade de controle calcula a distância e a velocidade relativa ao veículo à frente, podendo ainda detectar outros veículos em diferentes faixas da pista, de modo a obter um alcance lateral. Entretanto, o fabricante ressalta que o sensor não é capaz de detectar obstruções estacionárias, tais como a extremidade de um engarrafamento ou barreiras de proteção da pista (conhecidas por *guard rails*). Ao detectar a aproximação de outro veículo mais lento, o sistema diminui a velocidade do veículo atuando corretamente na unidade de gerenciamento do motor e, se necessário, no sistema de freios. Neste último caso, a atuação nos freios é limitada a 30% da capacidade máxima de frenagem do veículo, o que faz com que avisos luminosos e sonoros sejam emitidos ao motorista caso se faça necessária uma frenagem maior do que este valor (VOLKSWAGEN, 2016a).

Figura 21 – Ilustração do funcionamento do controle de cruzeiro adaptativo.

Fonte: [Volkswagen of The Woodlands \(2015\)](#).

Uma vantagem importante do sistema ACC é a sua contribuição para o aumento da segurança do veículo ao passo que o sistema atua permanentemente para manter uma distância segura ao veículo a frente, evitando assim possíveis acidentes ([VOLKSWAGEN, 2016a](#)). Reitera-se, por fim, que as vantagens do sistema CC também são válidas para o sistema ACC, dado que este corresponde a uma extensão do primeiro.

2.2.3 Sistema pré-colisão

O sistema pré-colisão (*Precrash Systems*) é capaz de detectar a possibilidade da ocorrência de um acidente e, neste caso, gera sinais de alerta ao motorista, podendo em alguns casos tomar ações preventivas de modo automático, como o fechamento eletrônico de janelas, tensionamento dos cintos de segurança ou até o pré-acionamento dos freios. Semelhantemente ao controle de cruzeiro adaptativo, o sistema pré-colisão pode utilizar informações provenientes de sensores montados à frente do veículo, como sensores *laser*, radares, ou outras formas de detecção com vídeo câmeras ([SHAOUT; COLELLA; AWAD, 2011](#)).

Um exemplo de sistema pré-colisão é o *Audi Pre Sense City* presente nos veículos Audi A4. Segundo o fabricante, este sistema, por meio de uma câmera montada no pára-brisa e com alcance de até 100 metros, é capaz de varrer a via em busca de pedestres ou veículos. Caso seja detectada a possibilidade de uma colisão, o condutor receberá diversos avisos e, se necessário, o sistema irá atuar nos freios, podendo acioná-los totalmente. Até 40 km/h, o *Audi Pre Sense City* é capaz de evitar por completo uma colisão. Entretanto, para velocidades maiores do que este valor e limitadas a 85 km/h, o sistema não evita o acidente, mas reduz o impacto da batida. Já para valores acima de 85 km/h, não existe atuação do sistema ([AUDI AG, 2016a](#)).

O sistema *Pre Sense Plus* da Audi, ilustrado na [Figura 22](#), é composto de até 4 estágios na sua versão completa. Os eventos de cada estágio e a ordem em que ocorrem estão destacados a seguir ([AUDI AG, 2016c](#)):

1. Emissão de alertas ao motorista;
2. Leve ativação dos freios do veículo, de modo a chamar a atenção do condutor;
3. Frenagem do veículo com aplicação de aproximadamente 1/3 da capacidade de frenagem máxima, além de provocar o fechamento de janelas e teto solar, tensionamento dos cintos de segurança, e ativação de luzes de emergência;
4. Frenagem mais acentuada do veículo, seguida de uma frenagem total, sendo que neste momento os cintos se encontram totalmente tensionados.

Figura 22 – O sistema *Audi Pre Sense Plus*.

Fonte: [Audi AG \(2016c\)](#).

Vale lembrar que até o segundo estágio ainda existe a possibilidade do condutor realizar uma manobra preventiva, de modo a evitar o acidente. No último estágio, a frenagem máxima ocorre por volta de meio segundo antes da colisão, a qual se torna inevitável nesta situação. Como resultado da aplicação deste sistema, a velocidade do veículo pode ser reduzida para um valor em torno de 40 km/h antes da batida, reduzindo consideravelmente as consequências do acidente ([AUDI AG, 2016c](#)).

2.2.4 Sistema de aviso de mudança de faixa

Ilustrado na [Figura 23](#), o sistema de aviso de mudança de faixa, conhecido também por *Lane Departure Warning System* (LDWS), é capaz de detectar as marcações da linha de separação das faixas em uma via, com objetivo de emitir alertas ao condutor em caso de mudança de faixa não intencional. Já quando a mudança de faixa é detectada como intencional, através do monitoramento de variações na direção e na aceleração do veículo, o sistema não emite nenhum alarme ([MAZDA MOTOR, 2016](#)). Além disto, é importante ressaltar que o sistema pode ser desativado pelo condutor através da interface HMI, omitindo assim todos os seus possíveis alertas ([SHAOUT; COLELLA; AWAD, 2011](#)).

Figura 23 – Ilustração do sistema *Lane Departure Warning System*.

Fonte: [Mazda Motor \(2016\)](#).

As formas de alerta ao motorista podem variar de acordo com a implementação do sistema. Em geral, são formadas por avisos sonoros ou visuais, mas em alguns sistemas estes alertas podem ser dados através de vibrações no volante, com intuito de simular possíveis irregularidades da pista para chamar a atenção do condutor ([SHAOUT; COLELLA; AWAD, 2011](#)). No sistema *Active Lane Assist* da Audi, por exemplo, esta forma de alerta é realizada através de motores de vibração montados no volante do veículo, como exibe a ilustração da [Figura 24](#).

Figura 24 – Motor de vibração utilizado pelo sistema *Audi Active Lane Assist* para alertar o motorista.

Fonte: [Audi AG \(2016b\)](#).

Neste sistema da Audi, uma câmera montada na parte frontal do espelho retrovisor é capaz de detectar as faixas da pista a uma distância de mais de 50 metros e ângulo de cobertura de 40°, fornecendo imagens de alta definição à taxa de 25 imagens por segundo. Vale lembrar que, de acordo com o fabricante, este sistema somente atua em velocidades acima de 60 km/h ([AUDI AG, 2016b](#)).

O sistema LDWS se mostra bastante eficiente sobretudo em situações em que se trafega por estradas predominantemente retas, pois neste caso existe uma tendência maior de desatenção do motorista em relação à estrada, o que poderia resultar em um acidente

(MAZDA MOTOR, 2016). Desta forma, o LDWS se mostra um componente ADAS de grande importância para o aumento da segurança do veículo.

2.2.5 Sistema de assistência a estacionamento

Em certas ocasiões, manobrar o veículo para estacioná-lo corretamente nas vias das cidades pode ser uma tarefa difícil e estressante para alguns motoristas (PSA GROUP, 2016). Desta maneira, sistemas capazes de auxiliar o motorista durante a realização de uma manobra de estacionamento se tornam elementos fundamentais para aumento do conforto do veículo dado que, neste caso, o motorista exercerá um esforço menor para a realização da tarefa. Estes sistemas, conhecidos por sistemas de assistência a estacionamento (*Parking Assistance Systems*), foram desenvolvidos pela primeira vez na metade dos anos 90, porém somente em 2003 passaram a ser produzidos em larga escala com o veículo Toyota Prius (SHAOUT; COLELLA; AWAD, 2011).

Em veículos Volkswagen, por exemplo, este sistema é capaz de estacionar o veículo mesmo em espaços apertados, sendo o motorista responsável somente pelo controle dos pedais. Para a realização de uma baliza, por exemplo, basta o condutor acionar o botão do sistema e seguir as instruções para parar o veículo próximo ao espaço da baliza. Uma vez que o veículo esteja parado corretamente no local, o condutor pode engatar a marcha ré e liberar o volante para ser controlado pelo sistema. A medida que o veículo se movimenta lentamente, sensores varrem a área ao seu entorno para determinar o procedimento correto da manobra. Além disto, o procedimento pode ser acompanhado através de um *display* que exibe ao motorista o caminho reverso do veículo. Apesar de o condutor não precisar assumir a direção, o fabricante ressalta que o mesmo tem pleno controle sobre o veículo ao trabalhar com o pedal do acelerador, freio e embreagem, além de ser possível desativar o sistema em qualquer instante por meio de atuação no volante ou quando a velocidade ultrapassa um valor limite (VOLKSWAGEN, 2016c).

Outro exemplo de sistema de assistência a estacionamento corresponde ao *Park Assist* desenvolvido pelo grupo PSA Peugeot Citroën, que já está presente nos veículos modelo Peugeot 208 e Citroën C4. Analogamente ao sistema da Volkswagen, quando requisitado pelo motorista, este sistema é capaz de identificar um espaço adequado para estacionamento e, ao encontrá-lo, controla a direção do veículo para realização correta da manobra no referido local. Apesar da necessidade do controle dos pedais pelo condutor, o fabricante afirma que a operação será totalmente automática na próxima geração do sistema, denominada *City Park Full Automatic* (Figura 25) e que está prevista para 2017. Ainda segundo o fabricante, o sistema será livre de estresse, uma vez que este será encarregado de gerenciar automaticamente direção, velocidade e mudança de marcha, ficando o condutor apenas como um supervisor da operação de manobra, capaz de retomar o controle do veículo a qualquer instante, se necessário (PSA GROUP, 2016).

Figura 25 – Sistema *City Park Full Automatic* do grupo PSA Peugeot Citroën, previsto para 2017.

Fonte: [PSA Group \(2016\)](#).

2.2.6 Sistema de detecção de sonolência

Acidentes causados por sonolência respondem por grande parte do total de acidentes que ocorrem todos os anos nas estradas ([SHAOUT; COLELLA; AWAD, 2011](#)). De acordo com dados do Observatório Nacional de Segurança Viária (ONSV), sonolência e cansaço são responsáveis por metade dos acidentes em rodovias brasileiras, o que evidencia a necessidade de o motorista redobrar a atenção ao dirigir no período noturno ([OBSERVATÓRIO NACIONAL DE SEGURANÇA VIÁRIA, 2015](#)).

Neste cenário, sistemas capazes de detectar um padrão de comportamento do motorista, respondendo de diversas maneiras, se tornam instrumentos importantes para a redução deste tipo de acidente ([SHAOUT; COLELLA; AWAD, 2011](#)). Estes sistemas são denominados por sistemas de detecção de sonolência, conhecidos na literatura também por *Drowsiness Detection Systems*.

O sistema *Driver Attention Alert* da Nissan, ilustrado na [Figura 26](#), é um exemplo de sistema de detecção de sonolência. Com base em informações de sensores presentes na direção do veículo, este sistema é capaz de estabelecer um padrão de dirigibilidade em cada instante de tempo, de modo fornecer dados a respeito da maneira como o motorista

está conduzindo o veículo. Caso o sistema detecte algum padrão indicativo de sonolência, alertas sonoros serão emitidos, além da exibição no painel do veículo de uma xícara de café juntamente com uma mensagem convidando o condutor a uma parada de descanso. Além disto, o sistema conta com a presença de algoritmos para lidar com falsas detecções, tais como mudanças de faixa, frenagens e condições adversas da via (NISSAN MOTOR, 2015).

Figura 26 – Exemplo de aviso emitido pelo sistema *Driver Attention Alert* da Nissan.

Fonte: Nissan Motor (2015).

2.3 A Comunicação dentro do Veículo

O termo ECU deixou de se referir apenas ao controle do motor, abrangendo também itens como segurança, conforto e dirigibilidade. Atualmente, estima-se que um carro mais sofisticado possua mais de 100 ECUs, cada uma realizando uma tarefa distinta no veículo (SHEDEED et al., 2013). Assim, se faz necessária a interligação destes módulos eletrônicos através de uma interface de comunicação adequada.

2.3.1 A interface CAN

A interface CAN, criada pela empresa alemã Bosch, corresponde a uma interface serial desenvolvida especialmente para interligar subsistemas eletrônicos (ECUs) de um carro, conforme ilustra o digrama da Figura 27 (BOSCH, 2005). Atualmente, devido à sua confiabilidade e robustez, esta interface não se restringe apenas ao ramo automotivo,

podendo ser empregada também em aplicações domésticas ou industriais ([FREESCALE, 2012](#)).

Figura 27 – Diagrama de um barramento CAN.

Fonte: [CAN Bus kit \(2016\)](#).

Dentre as características da interface CAN, destacam-se a sua capacidade para comportar vários mestres, interligação de diversos dispositivos com apenas dois fios, capacidade de se arbitrar o acesso simultâneo ao barramento adotando níveis de prioridade para as mensagens, flexibilidade na configuração, detecção e sinalização de erros, capacidade de retransmissão automática de mensagens corrompidas e taxa de transmissão de até 1 Mb/s ([ROBERT BOSCH GMBH, 1991](#)).

2.3.2 Sistema de diagnóstico de bordo - OBD

Durante os anos 70, os fabricantes de veículos deram início ao uso de meios eletrônicos para controlar funções do motor, de modo a atender normas que visavam o controle sobre as emissões de poluentes. Neste cenário, sensores envolvidos no controle eletrônico do motor, que forneciam informações ao sistema para a minimização dos

poluentes, passaram a ser acessados também para prover informações de diagnóstico (B&B ELECTRONICS, 2011).

No início desta prática, cada fabricante possuía seu próprio sistema de diagnóstico, seguindo um determinado protocolo estabelecido internamente por ele. Entretanto, em 1988, a Sociedade dos Engenheiros Automotivos (*Society of Automotive Engineers* - SAE) sugeriu a padronização de um sistema de diagnóstico, definindo um conector padrão e um conjunto de sinais de teste. Surgia então a primeira versão do sistema de diagnóstico de bordo (*On-Board Diagnostic* - OBD), conhecida também por OBD-I (B&B ELECTRONICS, 2011).

A primeira versão do OBD não era totalmente eficaz pelo fato de ser capaz de monitorar apenas um número limitado de elementos relacionados às emissões, além do fato de seus monitores não admitirem calibração para um determinado nível de emissão de poluentes. Desenvolve-se, então, a segunda versão do OBD (denominada OBD-II) para correção destes problemas, tornando o sistema mais robusto e de fácil manuseio (CALIFORNIA AIR RESOURCES BOARD, 2015).

Atualmente, existem no mercado diversos adaptadores OBD-II que podem ser adquiridos por um custo relativamente baixo, a exemplo do adaptador da Figura 28 que disponibiliza interface *bluetooth* para comunicação. Ao conectar este adaptador à porta OBD-II do veículo, o processo de diagnóstico e monitoramento pode ser realizado através de um aparelho celular que seja do tipo *smartphone*, tenha interface de comunicação *bluetooth* e tenha instalado um aplicativo móvel compatível com o respectivo adaptador, a exemplo do aplicativo *Torque* destacado na Figura 29.

Figura 28 – Adaptador OBD-II com comunicação *bluetooth*.

Fonte: Autor.

Figura 29 – Aplicativo *Torque* para uso com adaptador OBD-II.

Fonte: Ian J. Hawkins (2016).

Através de uma extensa lista de códigos⁵, é possível identificar e corrigir diversas falhas que possam ocorrer nos sistemas do veículo. Além disto, o sistema OBD-II também permite o monitoramento em tempo real de diversos parâmetros extraídos das ECUs, tais como rotação do motor, temperatura do motor, tensão da bateria, posição do pedal de aceleração, velocidade do veículo, dentre outros.

É importante ressaltar que o OBD-II se tornou obrigatório nos EUA em veículos fabricados a partir de 1996, enquanto no Brasil a obrigatoriedade começou a valer somente a partir de 2010 (BELLINI, 2014). Entretanto, existe a possibilidade de que veículos fabricados previamente ao ano de 2010 sejam parcialmente ou totalmente compatíveis com padrão OBD-II, a exemplo do Ford Fiesta Sedan 2009 1.6 Flex, cujo conector OBD-II pode ser localizado no painel abaixo do volante do veículo, como mostra a Figura 30.

⁵ A lista de códigos pode ser consultada em OBD-Codes.com (2016).

Figura 30 – Conector OBD-II de um Ford Fiesta Sedan 2009 1.6 Flex.

Fonte: Autor.

3 Metodologia para Aprimoramentos na ECU

O Projeto Otto IV propõe, conforme explicado no [Capítulo 1](#), o desenvolvimento de um *Cruise Control* para operar em um veículo Volkswagen Polo Sedan 2.0L 2004, sendo que para isto utiliza como plataforma de trabalho a unidade de gerenciamento eletrônico desenvolvida no âmbito do projeto Otto II ([PEREIRA, 2013](#)) para controle do motor presente neste veículo. Entretanto, previamente ao projeto do controle de cruzeiro, o trabalho, em sua primeira fase, tem como foco o aprimoramento desta unidade de gerenciamento eletrônico, cujo projeto ocorreu no ano de 2013.

Para isto, o *firmware* desenvolvido em 2013 passa por uma profunda mudança em sua estrutura para receber melhorias e novos recursos. Esta primeira fase do trabalho, portanto, envolve tanto atividades voltadas para o aperfeiçoamento de malhas de controle já existentes no *firmware* de 2013 (controle de marcha lenta, controle de posição da borboleta eletrônica, controle de partida, etc.), como também outras destinadas à implementação de novos recursos (controle de torque, controle da mistura ar/combustível em malha fechada, compensação de parâmetros de injeção e ignição, segurança na comunicação entre blocos da ECU, identificação da marcha engatada, suporte para diagnóstico do sistema via protocolo OBD-II, dentre outros).

Cita-se, a título de exemplo, a substituição do controle de rotação modelado no trabalho de 2013 por um controle de torque em malha aberta, ficando o primeiro restrito apenas ao regime de marcha lenta. Espera-se, por meio desta reformulação da ECU, uma melhora significativa na dirigibilidade do veículo, promovida pelo fato da entrada do sistema (pedal de aceleração) passar a ter uma relação de proporcionalidade maior com a aceleração do veículo, característica favorável também à implementação do controle de cruzeiro.

De modo a complementar o desenvolvimento de *firmware*, esta fase do trabalho envolve também o desenvolvimento de um novo *software* de monitoramento da ECU, que disponibiliza ao desenvolvedor do sistema diversos recursos, tais como a função de computador de bordo alternativo, opção para controle do motor através da simulação do pedal de aceleração, opção para alteração da rotação do regime de marcha lenta, função para automatização do ensaio de identificação do veículo (tarefa necessária para o projeto do controlador de cruzeiro, a ser discutida no [Capítulo 4](#)), dentre outros.

As seções deste capítulo abordam detalhes referentes às tarefas da primeira fase do trabalho, sendo que os resultados são apresentados na parte final deste capítulo. Além disto, a [seção 3.1](#) disponibiliza uma breve descrição do funcionamento da ECU desenvolvida em 2013.

Ressalta-se ainda que o [Apêndice A](#) deste trabalho disponibiliza para consulta a implementação em *firmware* dos principais aperfeiçoamentos realizadas na ECU do projeto, destacando, para isto, trechos do código embarcado escrito em linguagem C. O [Apêndice B](#), por sua vez, destaca os principais trechos da implementação da ferramenta de *software*, desenvolvida em linguagem C#.

3.1 Descrição da ECU de 2013

A unidade de gerenciamento eletrônico denominada ECU v2.0, cujo *hardware* foi projetado em 2013 no âmbito projeto Otto II, é composta por três principais blocos que dividem as tarefas do sistema, caracterizando uma arquitetura descentralizada com três unidades de processamento, como mostra o diagrama da [Figura 31](#).

Figura 31 – Diagrama de blocos do *hardware* da ECU v2.0.

Fonte: Autor (2013).

Nota-se ainda um grande fluxo de dados entre os processadores, o que é uma característica marcante da arquitetura descentralizada. Assim, para que o controle do motor funcione adequadamente é necessário que a comunicação entre os microcontroladores da placa ocorra de forma rápida e confiável.

As principais tarefas de cada bloco estão resumidas a seguir:

- Gerenciamento: Leitura de sensores e cálculo de parâmetros de atuação para injeção, ignição, válvula borboleta eletrônica e relés;
- Sincronismo: Geração dos comandos de atuação para injeção e ignição, com parâmetros recebidos do bloco de Gerenciamento via interface SPI;
- Comunicação: Coleta periódica de dados dos blocos de Gerenciamento e Sincronismo (via interface SPI), e envio dos mesmos para um computador externo (via interface UART/USB) e para o painel do motorista do veículo (via interface CAN), além da leitura de velocidade do veículo (via interface CAN).

O componente principal de cada bloco corresponde a um microcontrolador (μC) automotivo Freescale da família S12XE, dedicado para aplicações de gerenciamento de motores automotivos ([FREESCALE, 2013](#)). Operando a uma frequência de 50 MHz, este microcontrolador é capaz de processar um grande volume de dados de forma rápida e eficaz, além de disponibilizar ao programador diversos recursos implementados em *hardware*, tais como *timers* de alta resolução, controladores de comunicação serial (UART, SPI, I2C, CAN), conversores analógico/digital (A/D), dentre outros recursos exibidos no diagrama da [Figura 32](#).

Figura 32 – Diagrama de blocos da família de microcontroladores S12XE.

Fonte: [Freescale \(2013\)](#).

Dentre os recursos do S12XE, destaca-se o seu coprocessador denominado *XGate*. Composto de um processador RISC de 16 bits, o *XGate* é capaz de executar instruções em até 100 MHz e pode ser utilizado para atender diversas interrupções do microcontrolador, aliviando, portanto, a carga do seu processador principal S12X ([FREESCALE, 2012](#)). Deste modo, este coprocessador se torna uma ferramenta extremamente útil para o desenvolvimento de aplicações de gerenciamento de motores automotivos, dado que estas geralmente envolvem o processamento de um número elevado de interrupções.

O protótipo da ECU v2.0, exibido na [Figura 33](#), foi projetado com uso de componentes SMD de tamanho reduzido, incluindo resistores e capacitores, o que resultou em uma placa de circuito impresso relativamente pequena (15cm x 15cm) para a aplicação em questão, mesmo com a presença de três microcontroladores no circuito. Além disto, o custo de produção da placa de circuito impresso foi reduzido drasticamente ao se utilizar somente duas camadas de roteamento, sendo a camada inferior destinada quase que exclusivamente para aterramento¹. Por fim, destaca-se a disposição dos componentes apenas na camada superior da placa, o que facilita o processo de montagem do protótipo que, no trabalho

¹ A camada de aterramento tem como função auxiliar na blindagem da placa contra ruídos.

de 2013, foi realizado em sua maior parte manualmente com uso de ferro de solda, pinça, lente de aumento e estação de ar quente. Para maiores detalhes a respeito do *hardware* da ECU v2.0, recomenda-se a consulta à documentação do projeto Otto II ([PEREIRA, 2013](#)).

Figura 33 – Protótipo da ECU v2.0.

Fonte: Autor (2013).

As seções seguintes abordam, do ponto de vista de cada bloco do sistema, a estratégia de controle implementada na ECU de 2013. Detalhes do funcionamento do *firmware* poderão ser consultados no trabalho de 2013 ([PEREIRA, 2013](#)).

3.1.1 Bloco de Gerenciamento

O microcontrolador de Gerenciamento é responsável pelo cálculo dos parâmetros de atuação do sistema, tomando como base os valores dos sensores presentes no motor. No *firmware* desenvolvido no trabalho de 2013, este processamento é direcionado pelo controle de rotação, como mostra o diagrama da [Figura 34](#).

Figura 34 – Diagrama do controle de rotação do trabalho de 2013 (Projeto Otto II).

Fonte: Autor (2013).

Dada uma certa referência de rotação, o sistema atua simultaneamente no tempo de injeção de combustível e na abertura da válvula borboleta eletrônica para garantir o menor erro possível, de modo que a rotação do motor esteja o mais próximo possível da sua referência, esta que, por sua vez, é definida pelo motorista através do pedal de aceleração do veículo e pode variar de 800 RPM com pedal solto até 6500 RPM com pedal totalmente acionado. É importante notar também que a carga imposta ao motor atua como distúrbio no controle, de maneira a reduzir a sua rotação. Assim, cabe ao controle de rotação atuar de maneira a impedir esta redução, o que, conseqüentemente, fornece torque ao eixo do motor.

Como mostra o diagrama da [Figura 34](#), uma parte do controle de rotação é composto por um controlador puramente proporcional (P), cuja finalidade é enriquecer a mistura a/c quando a referência de rotação for maior que a rotação medida, situação em que o condutor exige uma aceleração do motor. Para esta situação, o controlador P fornece um ganho para o tempo de injeção estimado com base na massa de ar admitida pelo motor.

Além deste controlador P, o controle de rotação também é composto por outro controlador do tipo proporcional e integral (PI) que determina a abertura da válvula borboleta eletrônica, responsável pelo controle da massa de ar admitida pelo motor. Ganhos variáveis são adotados para este controlador, a fim de fornecer um controle que seja suave quando o pedal é pouco pressionado e, ao mesmo tempo, agressivo quando o pedal é bastante pressionado. Além disto, outros tratamentos estão embutidos neste controlador como, por exemplo, as limitações de abertura da borboleta para impedir o afogamento do motor por excesso de ar.

Outra malha presente no bloco de Gerenciamento corresponde ao controle de posição da válvula borboleta eletrônica. Com realimentação fornecida pelo sensor de posição TPS, esta malha de controle atua na tensão aplicada ao servomecanismo da borboleta eletrônica para garantir que a sua posição seja o mais próximo possível do valor de referência que, no caso do *firmware* de 2013, é determinado pelo controle de rotação descrito anteriormente. Vale mencionar, ademais, que este controle da borboleta eletrônica é implementado através de um controlador PI de ganhos fixos.

Os acionamentos dos bicos injetores e das bobinas de ignição são realizados pelo μC de Sincronismo, porém cabe ao Gerenciamento determinar os parâmetros adequados para que estes acionamentos ocorram corretamente. Para isto, este microcontrolador dispõe de tabelas armazenadas em memória não volátil que mapeiam os instantes em que os sinais de injeção e ignição devem ser desativados. Assim, a cada volta do motor estes valores são lidos pelo *firmware* de Gerenciamento, que então calcula os tempos de acionamento² e, em seguida, determina os parâmetros para acionar corretamente os respectivos sinais. Por fim, estes parâmetros são enviados ao μC de Sincronismo através do barramento SPI. Vale lembrar que este processo se repete a cada volta do motor.

Além da injeção, ignição e válvula borboleta, o bloco de Gerenciamento também é responsável pelo controle do relé da bomba de combustível. A estratégia de acionamento deste relé é bastante simples. Após o acionamento da linha 15 (chave da ignição), o Gerenciamento pressuriza a linha de combustível, ativando o relé por aproximadamente 0,5 segundo. Após isto, o relé é desligado e somente é religado após a primeira volta do motor.

3.1.2 Bloco de Sincronismo

Como mencionado anteriormente, o *firmware* do bloco de Sincronismo contém a estratégia para acionar precisamente os sinais de injeção e ignição, com base em parâmetros enviados pelo μC de Gerenciamento através da interface SPI. Para realização desta tarefa, além de receber corretamente os parâmetros do Gerenciamento a cada volta do motor, o

² O tempo de acionamento corresponde ao tempo de injeção, no caso da injeção, e ao tempo de carregamento da bobina, no caso da ignição.

μ C de Sincronismo mantém sincronia com o sinal de rotação, detectando e processando cada transição de subida deste sinal. Como este sinal está associado a posição de cada dente da roda fônica, pode-se afirmar que o Sincronismo realiza a contagem destes dentes para disparar os sinais no momento correto.

Para que a sincronia com o sinal de rotação seja mantida com sucesso, é importante que o Sincronismo, ao contrário do Gerenciamento, não execute muitos cálculos, uma vez que contas complexas, como multiplicação e divisão com ponto flutuante, levariam muito tempo para serem executadas e, conseqüentemente, alguns dentes poderiam não ser contabilizados (ou poderiam ser contabilizados com atraso). Este fato ainda se agravaria com o aumento da rotação, dado que o período de um dente diminuiria significativamente.

A título de exemplo, a [Figura 35](#) destaca os parâmetros de acionamento de um sinal de ignição. Durante a contagem dos dentes, o Sincronismo busca aquele que corresponde ao valor do dente de referência e, ao encontrá-lo, dispara um *timer* de alta resolução para medição dos tempos de disparo e de acionamento.

Figura 35 – Sinal de ignição com ilustração dos parâmetros de acionamento.

Fonte: Autor (2013).

Com relação aos modos de acionamento, para ignição se adota a centelha perdida, ou seja, os sinais de ignição são acionados simultaneamente em cada banco³, o que simplifica a estratégia de acionamento da ignição. Já a injeção é acionada apenas uma vez em cada cilindro, a cada duas voltas do motor, o que caracteriza a injeção multiponto sequencial. Vale lembrar que a sequência da injeção é mantida com auxílio do sensor de fase presente no motor utilizado no trabalho.

Por fim, é importante ressaltar que o bloco de Sincronismo presente no *hardware* da ECU v2.0 mantém correspondência direta com os *drivers* de sincronismo presentes na arquitetura de *software* de uma FlexECU, utilizada no desenvolvimento de ECUs comerciais. Isto se deve pelo fato de ambos os sistemas em questão (FlexECU e ECU v2.0) disponibilizarem recursos exclusivos (como, por exemplo, *timers*) para gerir corretamente o acionamento de injetores e bobinas de ignição, com base na leitura dos sinais dos sensores de rotação (*crankshaft*) e de fase (*camshaft*). Cita-se, a título de exemplo, o *driver* denominado *Engine Position Management* (EPM) que, presente na camada *Low Level Software* (LLSW) de uma FlexECU, é responsável por receber os sinais de rotação e de fase para sincronizar todo o *software* (SANTOS; ROSSETTI, 2015).

3.1.3 Bloco de Comunicação

O bloco de Comunicação é responsável por estabelecer a conexão entre a ECU e outros dispositivos externos, tais como computadores, aparelhos automotivos de diagnose e outras unidades eletrônicas presentes no veículo. Para isto, o microcontrolador deste bloco trabalha com auxílio de três interfaces de comunicação: UART, SPI e CAN.

O SPI é utilizado quando a comunicação ocorre internamente à ECU, ou seja, entre os blocos do sistema. Neste caso, a troca de dados necessita ocorrer de forma rápida, interferindo o menor tempo possível no processamento dos blocos de Gerenciamento e de Sincronismo, fato que justifica o uso de uma interface simples e rápida, como é o caso da interface SPI.

A interface UART, por sua vez, é utilizado para comunicação da placa com a aplicação de monitoramento do sistema, utilizando, para isto, um computador externo com sistema operacional Windows. Para facilitar conexões com computadores que não possuem porta serial nativa RS-232, o *hardware* da ECU v2.0 emprega um conversor UART/USB, responsável por realizar a conversão entre as interfaces de maneira totalmente transparente para o programador do sistema.

Por fim, a interface CAN é utilizada para troca de dados entre a ECU do motor e demais módulos presentes no veículo. Especificamente no projeto Otto II, a interface

³ Um banco é composto pelos cilindros 1 e 4, e o outro, pelos cilindros 2 e 3.

CAN é utilizada para se enviar o valor da rotação ao painel do motorista e para leitura da velocidade do veículo.

É importante ressaltar que o bloco de Comunicação opera continuamente, independente do regime do motor ou do estado da linha 15 (chave da ignição). Assim, o monitoramento ocorre normalmente mesmo quando o motor está desligado. Já o painel do motorista recebe informações via CAN somente com a linha 15 ligada (chave ligada no contato).

3.2 Controle de Torque em Malha Aberta

Como mencionado na [seção 3.1](#), no projeto de 2013 a entrada principal da ECU correspondia a uma referência de rotação, cujo valor era dado de acordo com a posição do pedal de aceleração do veículo, podendo variar de 800 RPM (pedal solto) até 6500 RPM (pedal totalmente pressionado). Uma vez determinado o valor de entrada, o controle principal da ECU atuava na válvula borboleta eletrônica para garantir que a rotação do motor fosse o mais próximo possível do valor de referência, inclusive em situações de carga.

Apesar do controle de rotação ter fornecido bons resultados em 2013, o presente projeto buscou outra estratégia para se controlar o motor a combustão interna, com intuito de se obter melhorias na dirigibilidade do veículo. Esta estratégia consiste em um controle de torque implementado, por simplicidade, em malha aberta. Desta forma, a entrada principal da ECU mantém correspondência com o pedal de aceleração do veículo, mas passa a ser compreendida como uma referência de torque relativo, cujo valor varia de 0 (pedal solto) até 100% (pedal totalmente pressionado).

Segundo [Kiencke e Nielsen \(2005\)](#), o controle de motores de combustão interna do tipo SI (*Spark Ignition*), em que a combustão se dá por meio de uma centelha, é realizado através da massa de ar admitida pelo motor, esta que, por sua vez, é controlada por meio da válvula borboleta montada no coletor de admissão. Neste trabalho, a abertura da borboleta eletrônica (α) é dada em função da referência de torque relativo (T_r), de acordo com a seguinte equação determinada empiricamente:

$$\alpha = (0,01 - 0,0001 \cdot \alpha_{base}) \cdot T_r^2 + \alpha_{base} \quad (3.1)$$

com $0 \leq T_r \leq 100$ e $0 \leq \alpha \leq 100$. A variável α_{base} , denominada neste trabalho por posição base, corresponde a um pequeno ângulo de abertura da borboleta necessário para o início do controle da marcha lenta. A [subseção 3.3.4](#) abordará mais detalhes referentes a esta variável.

É importante observar que a referência de torque máximo ($T_r = 100$) resulta na máxima abertura da borboleta ($\alpha = 100$), conforme mostra a curva da [Figura 36](#), a qual

representa o gráfico da equação 3.1. Em outras palavras, o torque máximo é esperado quando a borboleta de admissão está completamente aberta.

Figura 36 – Abertura da borboleta eletrônica em função da referência de torque relativo.

Fonte: Autor.

Ressalta-se ainda que a curva dada pela equação 3.1 foi determinada empiricamente por meio de testes no dinamômetro, com intuito de se obter melhorias em termos de dirigibilidade.

3.3 Controle da Rotação de Marcha Lenta

Conforme mencionado na subseção 2.1.4, o controle do regime de marcha lenta é muito importante para o funcionamento do motor a combustão interna, uma vez que pode auxiliar na economia de combustível e na redução de poluentes, principalmente em situações de congestionamento e trânsito intenso. Além disto, oscilações na rotação deste regime podem se tornar um forte incômodo para o condutor do veículo. Dada estas circunstâncias, o projeto Otto IV optou pela busca de uma solução que melhorasse significativamente o controle de rotação do regime de marcha lenta da ECU atual.

Para ativação do controle de marcha lenta, é necessário que a referência de torque seja menor do que 0,5%, condição que é atendida quando o pedal de aceleração está solto. Além disto, o controle somente começa a atuar quando a rotação cai abaixo de um certo valor denominado neste trabalho por rotação de controle, que corresponde à rotação de referência da marcha lenta acrescida de 300 RPM (ou 500 RPM quando a temperatura do motor estiver abaixo de 50 °C).

3.3.1 Primeira versão

A primeira versão do controle de marcha lenta desenvolvido no presente trabalho tomou como base o controle de rotação modelado em 2013, que era composto por um controlador PI de ganho proporcional K_p variável responsável por atuar diretamente na abertura da borboleta de admissão, conforme explicado na [subseção 3.1.1](#). A seguir foram resumidas algumas das modificações realizadas nesta malha de controle projetada em 2013:

- Simplificação da malha de controle através da adoção de um ganho K_p fixo que atua somente quando o erro de rotação for positivo⁴;
- Recalibração dos ganhos do controlador PI;
- Implementação de nova estrutura para *anti-windup*⁵ da ação integral do controle PI;
- Aperfeiçoamento do controle de posição da válvula borboleta⁶.

A [Figura 37](#) ilustra o diagrama de blocos da primeira versão do controle de rotação do regime de marcha lenta, em que é possível observar que, da mesma forma que no trabalho anterior, o controle da marcha lenta é realizado apenas por meio de atuação na válvula borboleta eletrônica. Neste caso, o controlador PI da marcha lenta determina a melhor abertura para a válvula borboleta eletrônica, valor este que é repassado para o seu controle de posição para garantir a abertura desejada.

⁴ Neste caso em que o erro de rotação é positivo, o controle deve agir para aumentar a rotação de marcha lenta.

⁵ *Anti-windup* é utilizado para evitar um aumento excessivo da ação integrativa do controle nas situações em que ocorre saturação do atuador.

⁶ A [seção 3.4](#) abordará maiores detalhes referente às melhorias desta malha de controle.

Figura 37 – Diagrama de blocos da primeira versão do controle da rotação de marcha lenta (sem atuação na ignição).

Fonte: Autor.

Verificou-se, por meio de experimentos, que a rotação de marcha lenta se torna extremamente sensível a pequenas variações na abertura da borboleta, principalmente em rotações abaixo de 900 RPM. Nesta situação, em que a borboleta se encontra com um ângulo de abertura muito pequeno, qualquer pequena variação na sua posição já é suficiente para provocar a instabilidade do regime de marcha lenta, como mostra a [Figura 38](#).

Figura 38 – Oscilações na rotação de marcha lenta quando se utiliza controlador PI de ganho K_i fixo para controle da borboleta eletrônica.

Fonte: Autor.

Desta forma, para contornar este problema o presente trabalho exigiu um esforço para melhorar o controle de abertura da válvula borboleta (VB) implementado em 2013 por meio de um controlador PI. A estratégia adotada no trabalho atual consiste no emprego de um ganho integral K_i variável para este regime, de modo a impedir que a posição da borboleta, medida pelo sensor TPS, caia abaixo do seu valor de referência, o que poderia derrubar a rotação de marcha lenta ou até mesmo causar o afogamento do motor. Este procedimento é realizado com auxílio do sinal do erro de posição e da posição atual da borboleta. De acordo com o fluxograma da [Figura 39](#), o controlador PI utilizará um ganho K_i maior para abrir rápido a VB quando a sua posição cair abaixo da referência, situação em que o erro de posição fornece um valor positivo. Já se o erro for negativo, o ganho

K_i será menor para garantir um fechamento suave e preciso do mecanismo. Vale lembrar que este procedimento de variar o ganho integral somente ocorrerá quando a posição da borboleta for suficientemente pequena para caracterizar o regime de marcha lenta, caso contrário o ganho K_i será fixo.

Figura 39 – Fluxograma para determinação do ganho K_i do controle de posição da borboleta.

Fonte: Autor.

Embora, como exibe a [Figura 40](#), a estratégia descrita tenha fornecido ótimos resultados em regime permanente, o tempo de resposta ainda era considerado elevado quando comparado ao desempenho do controle de marcha lenta da ECU de fábrica do motor, além do fato de a rotação ter apresentado pequenas oscilações no transitório. Para mitigação destes pequenos problemas restantes, uma nova malha de controle com atuação no avanço de ignição foi implementada para auxiliar a primeira malha que atua na borboleta eletrônica. Com esta nova malha de controle, implementada por meio de um controlador do tipo PD (proporcional mais derivativo), foi possível diminuir consideravelmente o tempo

de resposta da marcha lenta, além de auxiliar na sua estabilização durante os transitórios. A subseção 3.3.2 aborda detalhes desta versão final do controle de marcha lenta, que utiliza tanto borboleta eletrônica quanto o avanço de ignição para controle da rotação de marcha lenta.

Figura 40 – Rotação de marcha lenta quando se utiliza controlador PI de ganho K_i variável para controle da borboleta eletrônica.

Fonte: Autor.

3.3.2 Versão final

A Figura 41 mostra o digrama completo do controle de marcha lenta implementado na ECU deste trabalho. Nele é possível observar que o controle da rotação de marcha lenta é realizado por meio de um controlador PI, responsável pela atuação na posição da borboleta, em conjunto com um controlador PD, responsável pela atuação no avanço de ignição. Neste último caso, o avanço de ignição tabelado recebe uma correção para auxiliar no controle de marcha lenta, sendo o seu valor incrementado (maior avanço) quando se deseja elevar a rotação ou decrementado (maior atraso) quando se deseja diminuí-la. É importante ressaltar que o avanço de ignição tabelado apenas sofre esta correção durante o regime de marcha lenta e, para fins de segurança, o valor de correção determinado pelo

controlador é limitado para evitar problemas com detonação, dando assim uma margem de segurança para a operação do controle.

Figura 41 – Diagrama de blocos da versão final do controle da rotação de marcha lenta.

Fonte: Autor.

As Figuras 42 e 43 exibem, respectivamente, o fluxograma do controlador PI e do controlador PD, que compõem o controle da rotação de marcha lenta da ECU deste trabalho.

Figura 42 – Fluxograma do controlador PI da marcha lenta (atuação na válvula borboleta eletrônica).

Fonte: Autor.

Figura 43 – Fluxograma do controlador PD da marcha lenta (atuação no avanço de ignição).

Fonte: Autor.

3.3.3 Referência de rotação da marcha lenta

A referência de rotação do controle de marcha lenta é determinada em função da temperatura do motor, como mostra a curva da [Figura 44](#). Desta forma, a rotação é mantida em 1100 RPM quando o motor está frio (temperatura abaixo de 30°C), sendo reduzida linearmente conforme o motor vai esquentando, até manter 800 RPM quando a sua temperatura excede 50°C.

Figura 44 – Referência de rotação da marcha lenta em função da temperatura do motor.

Fonte: Autor.

Certos fatores, no entanto, podem alterar esta referência de rotação determinada em função da temperatura, tais como:

- **Acionamento do ar condicionado:** Limita inferiormente a referência em 900 RPM;
- **Veículo em movimento ou com embreagem acionada:** Limita inferiormente a referência em 1000 RPM;
- **Condição pré-partida:** Define a referência em 1100 RPM para a partida do motor;
- **Presença de erro de *hardware*⁷:** Limita inferiormente a referência em 1000 RPM;
- **Comando para alteração da referência de marcha lenta via *software*⁸:** Modifica a referência no intervalo de 600 a 1100 RPM, de acordo com o valor selecionado na aplicação.

3.3.4 Posição inicial da borboleta eletrônica para controle da marcha lenta

A posição inicial da borboleta eletrônica para controle da marcha lenta, denominada neste trabalho por posição base, é aquela que deve ser assumida como sinal de controle inicial para o controlador PI da marcha lenta. Em outras palavras, quando o controle de marcha lenta começa a atuar, a referência de posição da borboleta passa a ser corrigida pelo controlador PI a partir desta posição base. Determinada empiricamente, a posição

⁷ A detecção de erros de *hardware* será abordada na [subseção 3.8.11](#)

⁸ Maiores detalhes referentes a este comando de alteração da marcha lenta serão abordados na [subseção 3.7.6](#)

base da borboleta é dada em função da temperatura do motor, como mostra a curva da Figura 45.

Figura 45 – Posição base da borboleta eletrônica para controle da rotação de marcha lenta.

Fonte: Autor.

3.4 Controle de Posição da Válvula Borboleta Eletrônica

Em veículos equipados com válvula borboleta eletrônica, o controle da massa de ar admitida pelo motor é realizado diretamente pela ECU, fato que possibilita a implementação de diferentes estratégias para controle do motor a combustão interna. Entretanto, para que isto seja possível, é necessário que o controle da válvula borboleta eletrônica funcione corretamente, de modo a garantir que a sua abertura seja o mais próximo possível do seu valor de referência determinado pela estratégia da ECU.

Conforme mencionado na [seção 3.3](#), o regime de marcha lenta exige um controle preciso da válvula borboleta eletrônica, pois nesta situação a rotação do motor se torna extremamente sensível a pequenas variações na abertura do mecanismo. Assim, em busca de melhorar o regime de marcha lenta do motor Volkswagen 2.0L, o presente trabalho realizou algumas modificações no *firmware* de 2013:

- Nova configuração do conversor analógico/digital (A/D) para que as leituras dos sensores envolvidos (TPS e pedal do acelerador) sejam efetuadas com 12 bits, em vez de 8 bits, de modo a aumentar a precisão dos valores lidos;
- Nova configuração do sinal PWM de atuação para que o mesmo seja gerado com 16 bits, em vez de 8, de modo a possibilitar correções mais finas no controle de posição da borboleta eletrônica;

- Recalibração dos ganhos do controlador PI, de modo a diminuir o erro de regime e o tempo de resposta;
- Emprego de um ganho K_i variável para o regime de marcha lenta, conforme abordado na [seção 3.3](#);
- Aumento do período de amostragem de 1 para 10 milissegundos.

Embora as modificações acima tenham contribuído para uma melhora significativa do controle de marcha lenta, identificou-se a presença de oscilações na posição da borboleta quando a sua referência diminui bruscamente, situação que pode prejudicar a operação do motor. A [Figura 46](#) destaca estas oscilações, que podem ocorrer após o fechamento rápido da borboleta eletrônica no caso em que se utiliza o controlador PI.

Figura 46 – Oscilações na posição da borboleta eletrônica quando a sua referência reduz bruscamente (controlador PI).

Fonte: Autor.

Para correção deste problema, um termo derivativo foi incluído na malha de controle, resultando em um controlador do tipo PID (Proporcional Integral Derivativo). O ganho derivativo K_d foi determinado empiricamente para amortecer a resposta da borboleta,

mitigando assim qualquer oscilação no seu controle de posição, conforme mostra o gráfico da Figura 47.

Figura 47 – Ausência de oscilações na posição da borboleta eletrônica quando a sua referência reduz bruscamente (controlador PID).

Fonte: Autor.

A Figura 48 destaca o diagrama de blocos do controle de posição da borboleta eletrônica implementado na ECU deste trabalho.

Figura 48 – Diagrama de blocos do controle de posição da borboleta eletrônica.

Fonte: Autor.

3.5 Controle da Mistura Ar/Combustível (Controle Lambda)

O controle da mistura ar/combustível (a/c) é encarregado de dosar a quantidade adequada de combustível a ser injetada no motor, baseando-se em informações de sensores e também na estratégia de controle adotada pela ECU para o regime de operação em que se encontra o motor em um determinado instante. No presente projeto, o controle da mistura é dividido em duas partes: controle em malha aberta (estimação) e controle em malha fechada (correção).

O controle em malha aberta (*feedforward control*) tem como finalidade fornecer uma estimativa da massa de combustível a ser injetada no motor, determinando um tempo de injeção base calculado com auxílio de informações de sensores e com dados do bico injetor. Uma vez calculado, este tempo de injeção base é corrigido por fatores multiplicativos que são determinados por meio da temperatura do motor, da referência de torque e da tensão da bateria. A seguir estão resumidas as etapas para determinação do tempo de injeção, tomando por base o trabalho de 2013:

1. Cálculo da rotação do motor;
2. Estimativa da massa de ar admitida por cilindro com base na equação de estado dos gases ideais⁹;
3. Cálculo da massa de combustível a ser injetada de acordo com a relação estequiométrica de 12,7g de ar para 1g de gasolina com 25% de etanol (PEREIRA, 2015);
4. Correção da massa de combustível calculada através de um fator multiplicativo determinado empiricamente com auxílio da sonda lambda, de tal modo que a mistura a/c esteja o mais próximo possível da relação estequiométrica¹⁰;

⁹ Ou seja, assume-se que o ar se comporta como um gás ideal.

¹⁰ Esta correção é necessária uma vez que o trabalho fez uso da gasolina E27, que contém 27% de etanol, em vez da gasolina E25 com 25% de etanol.

5. Cálculo do tempo de injeção, com auxílio do fluxo do bico injetor;
6. Aplicação de um segundo fator multiplicativo com base no valor de referência de torque (regime do motor) e na temperatura do motor;
7. Aplicação de um terceiro fator multiplicativo para compensação de quedas na tensão da bateria.

Testes revelaram que os cálculos descritos nas etapas anteriores fornecem um tempo de injeção que resulta em uma mistura a/c muito próxima da relação estequiométrica, principalmente no regime de marcha lenta. Entretanto, correções com base no sensor lambda devem ser levadas em conta para que fatores externos, como a qualidade do combustível utilizado, não prejudiquem a estequiometria da mistura a/c. Desta forma, ao utilizar o sensor lambda para correção do tempo de injeção estimado pelo controle *feedforward*, o controle da mistura a/c passa a ser realizado em malha fechada (*feedback control*), o qual é referido neste trabalho também por controle lambda.

Conforme mencionado na [subseção 2.1.2](#), o controle em malha fechada da mistura a/c somente ocorre em regimes estáveis do motor. Desta forma, algumas condições devem ser consideradas pela ECU para ativação desta malha de controle. No caso deste trabalho, a ECU calcula a variação da rotação e verifica se o seu valor é suficientemente pequeno para ativação do controle lambda. Em outras palavras, a correção da mistura a/c somente ocorrerá por meio do sensor lambda quando a rotação do motor estiver constante, denotando um regime estável em que não existe aceleração acentuada. Já quando houver aceleração ou desaceleração, o controle da mistura será realizado somente em malha aberta, ou seja, o tempo de injeção será determinado somente por meio da massa de ar e de fatores de correção (temperatura do motor, referência de torque e tensão da bateria), sem correções do sensor lambda.

Além da variação da rotação, outros parâmetros são considerados pela ECU deste projeto para ativar o controle lambda:

- Rotação constante (variação menor que 150 RPM/s);
- Rotação entre 500 e 5000 RPM;
- Motor quente (temperatura do motor maior que 50 °C);
- Tempo para aquecimento da sonda lambda expirado;
- Tensão do sensor lambda abaixo de 2,5V, o que confirma o aquecimento do sensor;
- Ausência de erro de *hardware*.

A implementação do controle da mistura a/c em malha fechada deste trabalho se baseou no projeto desenvolvido por [Pereira \(2015\)](#). Nele, o autor projetou dois controladores diferentes, sendo um modelado com sonda lambda de banda estreita e outro com sonda lambda de banda larga. O projeto Otto IV optou pela escolha do primeiro modelo, dado que o veículo utilizado neste trabalho opera com a sua sonda lambda original de fábrica, que é do tipo banda estreita. A [Figura 49](#) apresenta o diagrama de blocos deste sistema com sonda lambda de banda estreita:

Figura 49 – Diagrama de blocos do controle da mistura a/c em malha fechada.

Fonte: [Pereira \(2015\)](#).

A [Figura 50](#) apresenta a resposta de uma sonda lambda de banda estreita. Nota-se que o sensor fornece uma tensão entre 0,8 e 1V para mistura rica, e uma tensão em torno de 0,1V para mistura pobre. Além disto, observa-se também que a tensão varia bruscamente quando a mistura passa de rica para pobre, e vice-versa, conferindo uma resposta com característica não linear ([PEREIRA, 2015](#)). Por fim, destaca-se o valor de tensão de 0,5V indicando mistura próxima da relação estequiométrica ($\lambda = 1$), fato que justifica a adoção do valor 0,5 como referência do controle lambda, como mostra o diagrama da [Figura 49](#).

Figura 50 – Resposta da sonda lambda de banda estreita.

Fonte: [Pereira \(2015\)](#).

O controlador $C(s)$ escolhido por [Pereira \(2015\)](#) para controle da mistura em malha fechada consiste em um controlador do tipo PI, em que o ganho K_p da parcela proporcional foi escolhido suficientemente pequeno para não prejudicar a estabilidade do motor, ao passo que o ganho K_i da parcela integral é determinado para compensar o atraso do sistema, de acordo com a relação ([PEREIRA, 2015](#)):

$$K_i = \frac{90}{n} \quad (3.2)$$

em que n corresponde à rotação do motor em RPM. A discretização de $C(s)$ realizada no trabalho de 2015 resultou na equação de diferenças dada a seguir ([PEREIRA, 2015](#)):

$$C(z) = 1 + (K_p + K_i \cdot T_s \cdot \frac{1}{z-1}) \cdot e(z) \quad (3.3)$$

em que $K_p = 0,01$, K_i é dado pela equação 3.2 e T_s corresponde ao período de amostragem do controle, que no caso foi escolhido igual a 2 voltas do motor, ou seja:

$$T_s = \frac{120}{n}(s) \quad (3.4)$$

Neste trabalho, algumas modificações foram realizadas no controle de mistura a/c projetado em 2015, com intuito de aprimorar a estabilização do motor em marcha lenta:

1. Redução do ganho K_p pela metade;
2. Redução do ganho K_i em aproximadamente 1/8 do seu valor;
3. Redução do período de amostragem pela metade, ou seja, a correção da mistura a/c passa ocorrer a cada volta do motor;
4. Implementação de controle de aquecimento da sonda lambda, a ser abordado na [subseção 3.5.1](#).

Além disto, este trabalho optou pela implementação do controlador PI na sua forma incremental, também conhecido por PI de velocidade, de acordo com a equação de diferenças dada a seguir ([ANGÉLICO, 2016](#)):

$$u(z) = u(z - 1) + K_p \cdot (e(z) - e(z - 1)) + K_i \cdot T_s \cdot e(z) \quad (3.5)$$

com $K_p = 0,005$, $K_i = \frac{12}{n}$ e $T_s = \frac{60}{n}$. O fluxograma da [Figura 51](#) mostra a implementação desta malha de controle, responsável pela controle da estequiometria da mistura em malha fechada. Nele se evidencia a presença de *anti-windup* para evitar problemas com a saturação do sinal de controle, que no caso corresponde a um fator de correção do tempo de injeção. Desta forma, o valor da correção fica limitada em 15%, tanto para acréscimo como para decréscimo do fator de injeção, dando margem para o sistema operar de forma segura.

Figura 51 – Fluxograma do controle lambda.

Fonte: Autor.

3.5.1 Controle de aquecimento da sonda lambda

O controle do aquecimento da sonda lambda é realizado por meio de um sinal PWM (*Pulse Width Modulation*) com ciclo de trabalho constante em aproximadamente 91% e período de 5,5 segundos. Em outras palavras, o relé do aquecedor da sonda lambda é ligado por 5 segundos e então desligado por mais 0,5 segundo, sendo este ciclo repetido continuamente enquanto o controle da mistura a/c estiver operando em malha fechada.

O fluxograma da [Figura 52](#) mostra a máquina de estados implementada para o controle do aquecimento da lambda. Observa-se que o aquecimento da lambda somente é ativado quando a temperatura do motor ultrapassa os 50 °C, valor adotado como limite

para consideração de estratégia de motor quente. Uma vez ativado o aquecimento, o *firmware* do Gerenciamento aguarda a tensão do sensor lambda cair abaixo de 2,5 Volts (V), para então liberar um contador de 20 segundos¹¹. Uma vez expirado este tempo, a lógica da ECU passa a considerar que o sensor atingiu a sua temperatura de operação, liberando, então, o controle da mistura para operar em malha fechada.

¹¹ Este procedimento foi determinado empiricamente para garantir que a ECU somente inicie o controle lambda após o sensor ter alcançado a sua temperatura de operação.

Figura 52 – Fluxograma do controle de aquecimento da sonda lambda.

Fonte: Autor.

3.6 Segurança na Comunicação entre Blocos da ECU

Como mencionado anteriormente, o bloco de Sincronismo é responsável pelo disparo correto dos sinais de injeção e ignição, necessários para comandar os bicos injetores e as bobinas de ignição do motor. No entanto, para executar esta tarefa com sucesso, o microcontrolador (μC) deste bloco necessita de parâmetros calculados pelo μC do bloco de Gerenciamento. Para isto, uma vez calculados, o Gerenciamento envia estes parâmetros ao Sincronismo a cada volta do motor, através da interface de comunicação SPI que, neste trabalho, opera a uma taxa de transmissão de 12,5 Mb/s.

A taxa de transmissão elevada é necessária para evitar atrasos no processamento do Sincronismo, o que poderia prejudicar o funcionamento do motor ao atrasar o disparo dos sinais de injeção e ignição. Entretanto, uma taxa de transmissão elevada torna a comunicação mais suscetível à ruídos provenientes de fontes externas (como o próprio motor), o que poderia acarretar na mudança de um ou mais bits em determinados parâmetros enviados no barramento. Neste caso, ao receber estes parâmetros corrompidos, o Sincronismo poderia gerar sinais incorretos para os atuadores, o que pode ser altamente prejudicial para o funcionamento do motor.

Apesar destes problemas em potencial, no ano de 2013 o projeto Otto II testou intensamente o funcionamento de sua ECU, não relatando nenhum problema relacionado a ruídos no barramento SPI. Mesmo assim, o projeto Otto IV decidiu pelo aumento da segurança deste canal de comunicação entre Sincronismo e Gerenciamento, com finalidade de reduzir drasticamente a possibilidade de ocorrer problemas no funcionamento do motor devido a um possível processamento de dados corrompidos.

Para detectar a ocorrência de erros na transmissão SPI, o *firmware* da ECU deste trabalho emprega um mecanismo denominado *Cyclic Redundancy Check* (CRC), que corresponde a um código de detecção de erro calculado com base no resto da divisão polinomial do conteúdo que se deseja transmitir ou armazenar. Devido à sua facilidade de implementação em *hardware* e *software*, este algoritmo se tornou amplamente utilizado em dispositivos digitais tal como, por exemplo, o disco rígido de computador. Além disto, o CRC se revela como uma poderosa ferramenta para uso em comunicação digital ao passo que é capaz de detectar erros em um número arbitrário de bits de informação, contando ainda com uma taxa de detecção de erro de 99,9984% no pior caso ([FELICE, 2007](#)).

Os itens a seguir descrevem o uso do CRC para detecção de erros na transmissão de dados entre os blocos Gerenciamento e Sincronismo:

1. Com a definição dos dados a serem enviados ao Sincronismo, o Gerenciamento calcula o CRC aplicando um algoritmo nestes dados.

2. Uma vez calculado, este CRC passa a integrar o pacote de dados a ser enviado ao outro microcontrolador.
3. O μC do bloco de Sincronismo, ao receber os dados, separa o CRC calculado pelo Gerenciamento do restante dos dados, para então calcular outro CRC aplicando o mesmo algoritmo utilizado no μC de Gerenciamento.
4. Espera-se que este CRC calculado pelo Sincronismo seja exatamente igual ao CRC enviado pelo Gerenciamento, indicando que os dados não foram corrompidos durante a transmissão SPI. Se assim for, o Sincronismo prossegue de forma segura com a atualização de suas variáveis internas.
5. Entretanto, se os valores dos CRCs não forem iguais, o Sincronismo não atualizará os valores de suas variáveis interna e, neste caso, os parâmetros da volta serão idênticos aos parâmetros da volta anterior do motor, procedimento que não traz prejuízos para a sua operação. Além disto, o Sincronismo também ativará a sua *flag* de erro de *software*, para que o usuário tenha conhecimento da falha ocorrida naquela volta do motor.

A Figura 53 ilustra o procedimento descrito.

Figura 53 – Ilustração do uso de CRC para detecção de erros na transmissão de dados entre os blocos Gerenciamento e Sincronismo.

Fonte: Autor.

Atualmente, existem diversas variantes para o cálculo do código CRC de 16 bits, como se verifica em [Cook \(2016\)](#). Este projeto adotou o algoritmo denominado *CRC CCITT Kermit*, porém inicializado com valor *0xFFFF*, o que faz com que o mesmo seja conhecido também por *CRC CCITT False*. A implementação deste algoritmo tomou por base o algoritmo fornecido por [Felice \(2007\)](#), modificado de acordo com a parametrização do algoritmo de Kermit. A validação do algoritmo deste projeto, destacado no trecho de código a seguir, foi realizada por meio da ferramenta fornecida por [Lammert Bies \(2015\)](#).

```
// Rotina para calcular o CRC16 (algoritmo CCITT - variante "0
// xFFFF")
unsigned int calcula_crc16(unsigned char *data_p, unsigned int
length)
{
    unsigned char bit;
    unsigned int index, carry;
    unsigned int crc = 0xFFFF;
    static const unsigned int poly = 0x1021;

    for (index = 0; index < length; index++)
    {
        crc ^= ((unsigned int)data_p[index] << 8);
        for (bit = 0; bit < 8; bit++)
        {
            carry = crc & 0x8000;
            crc <<= 1;
            if (carry)
                crc ^= poly;
        }
    }
    return crc;
}
```

3.7 Nova Aplicação de Monitoramento

Com base na aplicação desenvolvida em 2013, o presente trabalho desenvolveu, em linguagem C#, uma nova aplicação de monitoramento da ECU. Denominada ECU v2.0 Monitor e concebida para rodar em qualquer computador equipado com o sistema operacional Windows, esta aplicação, exibida na [Figura 54](#), é capaz de monitorar diversos parâmetros do sistema, tais como valores de sensores e parâmetros de atuação calculados pela ECU deste trabalho.

Figura 54 – Aplicação de monitoramento ECU v2.0 Monitor.

Fonte: Autor.

Além disto, esta aplicação não se restringe apenas ao monitoramento da ECU do motor, dado que possui capacidade para ler e exibir na tela diversos parâmetros presentes no barramento CAN do veículo, tais como estado de luzes do painel do motorista (luz de bateria, freio de estacionamento, setas, pisca alerta, luz de falha da ECU do motor denominada EPC - *Electronic Power Control*), estado das portas do veículo, valor do hodômetro, nível de combustível, estado de acionamento do ar condicionado, dentre outros. Para a realização desta tarefa, no entanto, é necessário que o *firmware* do bloco de Comunicação esteja preparado para realizar a leitura destes parâmetros do barramento CAN, retendo estas informações em sua memória para disponibilizá-las à aplicação de monitoramento quando for requisitado.

O *software* desenvolvido se mostrou bastante útil no que diz respeito à verificação e depuração do sistema. Ao se constatar, por exemplo, uma falha durante os testes no veículo, é possível, pela interface, mapear todos os parâmetros que foram medidos e/ou calculados no momento em que ocorreu o problema, facilitando a compreensão, reprodução e correção da falha em laboratório. Para isto, a aplicação disponibiliza uma opção para registro em arquivo dos dados monitorados, o que viabiliza também a confecção de tabelas e gráficos com os parâmetros retirados do sistema.

As seções seguintes destacam os principais recursos da aplicação de monitoramento desenvolvida neste trabalho.

3.7.1 Taxa de amostragem flexível

Um dos principais recursos implementados no *software* de monitoramento consiste na definição do período de amostragem dos dados lidos da ECU. Conforme destacado na Figura 55, o usuário conta com a opção de diversos períodos de amostragem já pré-definidos na interface, sendo 100 milissegundos o valor padrão da aplicação.

Figura 55 – Seleção do período de amostragem na aplicação de monitoramento.

Fonte: Autor.

A escolha do período de amostragem se mostra uma ferramenta bastante útil para diversos experimentos, principalmente naqueles que envolvem aquisições para posterior processo de modelagem empírico, a fim de viabilizar o projeto de controladores para determinadas tarefas da ECU. Entretanto, para que isto seja possível, é necessário que a aplicação seja robusta o suficiente para garantir uma taxa de amostragem precisa e constante, tarefa esta que se complica quando o sistema operacional do computador não é de tempo real.

Para contornar este problema, o controle do período de amostragem na aplicação desenvolvida foi realizado por meio de um *timer* especial denominado *Multimedia Timer*. Diferentemente dos demais *timers* disponíveis para uso em C# no Windows, este *timer* tira todo proveito do *hardware* do computador para fornecer uma resolução de até 1 milissegundo (MICROSOFT, 2016a), valor suficiente para as finalidades deste trabalho.

Para conferir que o período de amostragem está de acordo com o valor selecionado pelo usuário, a aplicação salva no arquivo de *log* o registro de tempo de cada ciclo de

aquisição da ECU. Desta forma, após realizar toda a aquisição necessária, o usuário pode conferir no arquivo salvo que a taxa de amostragem definida por ele foi mantida constante ao longo do ensaio. Para implementação deste registro de tempo, a aplicação utiliza em seu código fonte a classe *Stopwatch*. Segundo [Microsoft \(2016b\)](#), esta classe utiliza os tiques de contadores internos do sistema para fornecer um contador de alta resolução, que pode ser utilizado pelo programador para medir o tempo de execução de trechos de códigos da aplicação. A [subseção 3.7.8](#) abordará mais informações a respeito do arquivo de *log*.

Vale mencionar que, neste trabalho, o menor período de amostragem que a aplicação é capaz de realizar um ciclo completo de aquisição de dados sem erros corresponde a 50 milissegundos, sendo que, para isto, a interface UART deve operar a uma taxa de 125000 b/s para transmitir 55 *bytes* de informação do μ C de Comunicação.

3.7.2 Computador de bordo alternativo

Outro recurso exclusivo da aplicação deste trabalho consiste no cálculo de parâmetros de computador de bordo como, por exemplo, o consumo médio e instantâneo do motor. Este recurso pode ser extremamente útil para projetos futuros que possam visar, por exemplo, a diminuição do consumo de combustível do veículo. Ao mesmo tempo, estes parâmetros calculados podem ser importantes do ponto de vista do usuário, principalmente para aqueles que possuem veículos que não dispõem de computador de bordo no seu estado de fábrica. Vale lembrar que o veículo utilizado neste trabalho possui computador de bordo simples, que consiste em um *display* que exibe apenas o consumo instantâneo do motor, na unidade km/L.

Após realizar um ciclo de leitura de parâmetros da ECU, a aplicação calcula os seguintes parâmetros de computador de bordo, destacados na [Figura 56](#):

1. Consumo instantâneo em km/L (ou L/h, se a velocidade for nula);
2. Consumo médio em km/L;
3. Autonomia em km;
4. Combustível consumido em L.

Figura 56 – Parâmetros de computador de bordo calculados pela aplicação de monitoramento.

Fonte: Autor.

O cálculo do consumo instantâneo é realizado com auxílio dos seguintes parâmetros da ECU: tempo de injeção, rotação, fluxo do bico injetor (constante), densidade do combustível (constante) e velocidade do veículo. Uma vez calculado, o consumo instantâneo é utilizado para atualizar o consumo médio, cujo valor é retido na memória da aplicação. A autonomia, por sua vez, é estimada com base no consumo médio e no nível de combustível lido da ECU. Por fim, o combustível consumido, cujo valor também é retido na memória da aplicação, é atualizado periodicamente com base no consumo instantâneo do motor.

3.7.3 Nova interface gráfica

As Figuras 57 e 58 exibem, respectivamente, as interfaces gráficas da aplicação de 2013 e a atual. A interface gráfica do *software* de monitoramento da ECU deste trabalho foi desenvolvida com intuito de se aproximar o máximo possível do painel do veículo utilizado para testes, no caso um Polo Sedan 2004. Como resultado, diversos objetos gráficos, como ponteiros e “luzes”, foram inseridos no *layout* da aplicação, proporcionando ao usuário uma familiaridade maior com o painel do motorista, além de facilitar a visualização de determinados parâmetros como, por exemplo, a rotação do motor, a sua temperatura e a velocidade do veículo.

Figura 57 – Interface gráfica da aplicação de monitoramento de 2013 (projeto Otto II).

Fonte: Autor (2013).

Figura 58 – Interface gráfica da aplicação de monitoramento (projeto Otto IV).

Fonte: Autor.

Alguns recursos gráficos, entretanto, foram adicionados à interface da aplicação com objetivo apenas de melhorar a sua apresentação, pois exibem informações que não estão presentes no painel do veículo. Um exemplo deste caso corresponde ao *display* de 7 segmentos que exibe a marcha engatada.

Outro recurso gráfico interessante que melhora a apresentação da aplicação consiste na alteração da cor do plano de fundo de acordo com o acionamento da lanterna do veículo. Esta cor passa de cinza para azul claro quando a lanterna do veículo é acionada, retornando para cinza quando a mesma é desligada pelo condutor. A [Figura 54](#) mostra uma situação em que a lanterna do veículo está apagada, ao passo que a [Figura 56](#) exibe um exemplo em que a mesma está acionada. Vale lembrar que o acionamento do farol baixo ou alto provoca na aplicação o mesmo comportamento do acionamento da lanterna.

3.7.4 Controle do pedal de aceleração pela aplicação de monitoramento

Além da função de monitoramento da ECU, a aplicação desenvolvida disponibiliza, somente para fins de teste, uma opção para controle do motor através da simulação do pedal de aceleração do veículo, como destacado na [Figura 59](#). Neste caso, quando o desenvolvedor seleciona a opção “Controlar Pedal”, a ECU corta o pedal de aceleração físico do veículo e passa a responder às variações do controle deslizante presente na aplicação. Este controle deslizante, por sua vez, fornece uma medida que varia de 0 (pedal solto) a 255 (pedal totalmente pressionado), que é então convertida pelo bloco de Gerenciamento em um valor de referência de torque, resultando, portanto, em um valor de 0 a 100%.

Figura 59 – Controle do pedal de aceleração pela aplicação de monitoramento.

Fonte: Autor.

Este recurso para se controlar o motor diretamente de um computador se mostra bastante útil em testes práticos, principalmente naqueles em que se necessita que o condutor mantenha o pedal de aceleração pressionado constantemente em uma certa posição, o que não é uma tarefa fácil. Enfatiza-se, entretanto, que este procedimento não é destinado para uso final pelo usuário, devendo ser utilizado apenas para testes por pessoas experientes.

Como medida de segurança, o *firmware* da ECU força o retorno do controle do motor para o pedal de aceleração do veículo sempre que ocorrer o acionamento do pedal da embreagem. Desta forma, caso ocorra algum problema com a aplicação ou com o computador em que a mesma esteja rodando, o condutor poderá acionar a embreagem para desarmar o controle da aplicação, se assim julgar necessário. Por esta razão, é importante que este procedimento de se controlar o motor pela aplicação seja realizado na presença de pelo menos duas pessoas, sendo uma responsável pelo controle da aplicação no computador, enquanto a outra, no exercício da função de motorista, é responsável pela atuação na embreagem, no caso de uma possível emergência.

Para reforçar ainda mais a segurança da ECU, o condutor do veículo conta com um botão de emergência que pode ser acionado a qualquer momento para desligar de imediato todos os atuadores do motor. Sendo assim, este botão pode ser utilizado como uma segunda opção para desativar o controle da aceleração pela aplicação, caso o acionamento

da embreagem falhe por algum motivo. A [subseção 3.8.12](#) abordará mais detalhes em relação ao botão de emergência adicionado à ECU v2.0.

Vale lembrar que este recurso para controle do motor a partir da aplicação já estava presente no *software* desenvolvido em 2013. Entretanto, naquele caso os valores limites do controle deslizante eram dependentes dos valores mínimo e máximo do sensor do pedal de aceleração do veículo (PEREIRA, 2013), o que representa uma desvantagem em relação ao *software* atual, que dispensa o conhecimento destes valores de calibração.

3.7.5 Seleção do modo de operação da ECU

A ECU v2.0 conta com três modos de operação, destacados na [Figura 60](#):

1. **Normal** (padrão do *firmware*): Neste modo, a ECU funciona de maneira análoga a ECU original do veículo, liberando abertura máxima (100%) da válvula borboleta quando requisitado pelo condutor (ref. de torque = 100%) e filtrando variações bruscas na posição da borboleta quando o veículo estiver engatado¹²;
2. **Econômico**: Ao ser selecionado, este modo divide a equação 3.1 pela metade, de modo que a abertura máxima da borboleta fica limitada em 50% quando o pedal de aceleração for totalmente pressionado (ref. de torque = 100%). Como resultado, o motor passa a funcionar com aberturas menores na borboleta, o que resulta em uma maior economia de combustível, apesar de resultar, ao mesmo tempo, em uma perda de potência;
3. **Esportivo**: Este modo de operação corresponde ao modo normal, porém sem o filtro na referência de posição da borboleta eletrônica, o que resulta em uma resposta mais rápida do motor em relação ao pedal de aceleração nas situações em que o veículo se encontra engatado. Entretanto, se as variações impostas pelo condutor no pedal forem acentuadas, poderão surgir “solavancos” no veículo, comportamento que pode gerar desconforto para os passageiros. Desta maneira, esta opção tem como intuito demonstrar, de forma prática, a importância do filtro da válvula borboleta eletrônica quando o veículo está engatado, ao mesmo tempo que permite uma resposta mais rápida do motor no que diz respeito à aceleração do veículo.

¹² A [subseção 3.8.7](#) abordará mais detalhes em relação a este filtro da borboleta.

Figura 60 – Seleção do modo de operação da ECU na aplicação de monitoramento.

Fonte: Autor.

Uma vez selecionado na aplicação, um comando específico é enviado ao bloco de Comunicação, que então repassa a informação ao Gerenciamento para que o comportamento do controle principal da ECU seja alterado conforme a descrição anterior. Vale lembrar que, uma vez alterado, o modo de operação fica retido na memória da ECU e não sofre alteração, mesmo com o *reset* da linha 15 (chave da ignição). Sendo assim, para retornar o modo de operação para o seu valor padrão (modo normal), é necessário selecioná-lo na aplicação ou então religar a energia da placa da ECU para resetar a configuração retida em memória RAM.

3.7.6 Seleção da referência de rotação do regime de marcha lenta

Inicialmente, a referência de rotação do regime de marcha lenta era definida exclusivamente pelo *firmware* da ECU v2.0, e o seu valor podia variar de 1100 RPM com motor frio (temperatura abaixo de 30 °C) até 800 RPM com a temperatura do motor acima de 50 °C (estratégia de motor quente). Entretanto, ao término do desenvolvimento do controle de marcha lenta, decidiu-se pela implantação de um comando no *firmware* da ECU capaz de alterar a rotação de marcha lenta, com intuito de demonstrar e validar o controle de rotação deste regime do motor.

A [Figura 61](#) destaca a localização do objeto gráfico disponível na aplicação para alteração da rotação de marcha lenta do motor. Com auxílio das setas, o usuário pode

elevar ou abaixar a rotação da marcha lenta em passos de 50 RPM, na faixa de 600 até 1100 RPM. Ressalta-se que este recurso é exclusivo da ECU deste trabalho, dado que este procedimento não é possível de ser realizado por meio da ECU original do veículo no seu estado de fábrica.

Figura 61 – Seleção da referência de rotação da marcha lenta na aplicação de monitoramento.

Fonte: Autor.

3.7.7 Leitura das versões de *firmware* dos blocos da ECU

Quando a placa da ECU é alimentada, o bloco de Comunicação requisita aos blocos de Gerenciamento e Sincronismo as suas respectivas versões de *firmware*, para então armazená-las em sua memória. Desta forma, quando o usuário clica no botão “Ler Versão Fw” em destaque na Figura 62, um comando é enviado pela interface USB/UART ao bloco de Comunicação, que então responde com as versões de *firmware* retidas em sua memória RAM. Vale lembrar que o bloco de Comunicação também retém em memória a sua própria versão de *firmware*.

Figura 62 – Leitura das versões de *firmware* na aplicação de monitoramento.

Fonte: Autor.

Para que o controle de versão seja realizado com sucesso, cabe ao desenvolvedor evoluir a versão de *firmware* de cada bloco quando houver qualquer mudança efetiva no comportamento da ECU. Além disto, vale lembrar que as versões de *firmware* de cada bloco são salvas no arquivo de *log* da aplicação, se o usuário requisitá-las antes do início da leitura dos dados da ECU¹³. Mais detalhes à respeito do arquivo de *log* serão abordados na [subseção 3.7.8](#).

3.7.8 Registro de parâmetros em arquivo (*Log*)

Da mesma forma que no projeto de 2013, a aplicação deste projeto foi desenvolvida com capacidade para salvar os parâmetros lidos da ECU. Entretanto, além de salvar um número maior de parâmetros em arquivo de texto, a aplicação atual ainda salva outras informações relevantes para o usuário, tais como:

- Data e hora do início da tomada de dados;
- Versões de *firmware* da ECU;
- Versão da aplicação de monitoramento;

¹³ Em outras palavras, o usuário deve clicar no botão “Ler Versão Fw” antes de clicar no botão “LER DADOS” para que as versões de *firmware* sejam salvas no *log*.

- Modo de operação da ECU;
- Temperatura do motor no início da aquisição;
- Período de aquisição (ou período de amostragem);
- Tipo de aquisição.

A Figura 63 exibe o botão presente na aplicação para salvar os dados da ECU em arquivo.

Figura 63 – Registro em arquivo dos parâmetros lidos pela aplicação de monitoramento.

Fonte: Autor.

A Figura 64 mostra um exemplo de arquivo de *log* obtido com a aplicação de monitoramento. Nota-se que o início do arquivo é formado por um cabeçalho com as informações citadas anteriormente. Em sequência a este cabeçalho, insere-se uma linha com a identificação de cada parâmetro lido da ECU e, em seguida, adicionam-se os dados da aquisição. É importante mencionar que cada linha do arquivo compreende a um ciclo de leitura, sendo cada parâmetro separado por um ponto e vírgula (;), de modo a facilitar a importação dos dados em ferramentas de trabalho como, por exemplo, o MATLAB ou o Excel (Figura 65).

Figura 64 – Arquivo de *log* da aplicação de monitoramento.

```

mlenta_carga_eletrica.txt - Notepad
File Edit Format View Help
Monitoramento e Diagnóstico - ECU v2.0
Data: 03/09/2016
Hora: 10:53
Versão da aplicação: 2.7.3.0
Firmware Gerenciamento: 1.6.3
Firmware Sincronismo: 1.3.1
Firmware comunicação: 1.3.8
Modo de operação: Normal
Temperatura do Motor: 79,71°C
Período de aquisição: 100 ms
Tipo de aquisição: Normal (tudo)

t_Aquisicao(ms); RefTorque(%); DeltaRPM; RPM; MlentaRPM; RefVB; VB; PWM_VB(%); Cruise; RefVeloc(km/h); veloc(km/h); velocCAN(km/h); Pedal; MAP; Tar; Tagua; L
99; 0,00; -15; 742; 750; 520; 516; 18,13; 0; 0; 0,00; 0; 641; 603; 100; 59; 80 (Rico); 255; solto; solto; Neutro; 0; 9; 1,17; 8,99; 81; 1,17; 3,34; 5,80; 9;
199; 0,00; -18; 747; 750; 520; 518; 13,53; 0; 0; 0,00; 0; 638; 563; 101; 59; 81 (Rico); 247; solto; solto; Neutro; 0; 9; 1,18; 8,99; 81; 1,17; 3,34; 5,80; 9;
299; 0,00; -26; 747; 750; 520; 522; 10,72; 0; 0; 0,00; 0; 639; 609; 101; 59; 114 (Rico); 253; solto; solto; Neutro; 0; 9; 1,17; 8,99; 81; 1,17; 3,32; 5,80; 9;
399; 0,00; -43; 733; 750; 520; 523; 11,04; 0; 0; 0,00; 0; 631; 677; 101; 59; 115 (Rico); 255; solto; solto; Neutro; 0; 9; 1,16; 8,99; 81; 1,17; 3,34; 5,80; 9;
499; 0,00; -60; 721; 750; 521; 523; 11,03; 0; 0; 0,00; 0; 639; 651; 101; 58; 121 (Rico); 253; solto; solto; Neutro; 0; 9; 1,17; 8,99; 81; 1,17; 3,45; 5,80; 9;
599; 0,00; -20; 723; 750; 521; 524; 11,76; 0; 0; 0,00; 0; 639; 627; 101; 59; 77 (Rico); 255; solto; solto; Neutro; 0; 9; 1,19; 8,99; 81; 1,17; 3,48; 5,80; 9;
699; 0,00; -18; 736; 750; 521; 521; 10,82; 0; 0; 0,00; 0; 640; 618; 100; 59; 67 (Rico); 249; solto; solto; Neutro; 0; 9; 1,22; 8,99; 81; 1,17; 3,52; 5,80; 9;
799; 0,00; -34; 740; 750; 521; 520; 10,96; 0; 0; 0,00; 0; 641; 655; 100; 59; 69 (Rico); 254; solto; solto; Neutro; 0; 9; 1,23; 8,99; 81; 1,17; 3,52; 5,80; 9;
899; 0,00; -16; 752; 750; 521; 524; 11,83; 0; 0; 0,00; 0; 640; 615; 101; 59; 63 (Rico); 249; solto; solto; Neutro; 0; 9; 1,25; 8,99; 81; 1,17; 3,52; 5,80; 9;
999; 0,00; 15; 756; 750; 521; 522; 12,26; 0; 0; 0,00; 0; 641; 655; 100; 59; 69 (Rico); 254; solto; solto; Neutro; 0; 9; 1,26; 8,99; 81; 1,17; 3,54; 5,80; 9;
1099; 0,00; 1; 748; 750; 521; 523; 12,31; 0; 0; 0,00; 0; 638; 609; 101; 59; 68 (Rico); 252; solto; solto; Neutro; 0; 9; 1,24; 8,99; 81; 1,17; 3,52; 5,80; 9;
1199; 0,00; -3; 744; 750; 521; 521; 12,77; 0; 0; 0,00; 0; 641; 670; 100; 59; 69 (Rico); 255; solto; solto; Neutro; 0; 9; 1,24; 8,99; 81; 1,17; 3,52; 5,80; 9;
1299; 0,00; -9; 738; 750; 521; 522; 11,90; 0; 0; 0,00; 0; 639; 605; 101; 59; 67 (Rico); 254; solto; solto; Neutro; 0; 9; 1,23; 8,99; 81; 1,17; 3,52; 5,80; 9;
1399; 0,00; 26; 759; 750; 521; 520; 11,55; 0; 0; 0,00; 0; 640; 661; 101; 59; 68 (Rico); 254; solto; solto; Neutro; 0; 9; 1,26; 8,99; 81; 1,17; 3,51; 5,80; 9;
1499; 0,00; 23; 744; 750; 521; 520; 11,16; 0; 0; 0,00; 0; 641; 610; 100; 59; 67 (Rico); 254; solto; solto; Neutro; 0; 9; 1,24; 8,99; 81; 1,17; 3,52; 5,80; 9;
1599; 0,00; 36; 759; 750; 521; 523; 10,43; 0; 0; 0,00; 0; 638; 665; 101; 59; 67 (Rico); 254; solto; solto; Neutro; 0; 9; 1,26; 8,99; 81; 1,17; 3,52; 5,80; 9;
1699; 0,00; 22; 758; 750; 521; 520; 11,20; 0; 0; 0,00; 0; 640; 607; 101; 59; 66 (Rico); 254; solto; solto; Neutro; 0; 9; 1,25; 8,99; 81; 1,17; 3,51; 5,80; 9;
1799; 0,00; 22; 762; 750; 521; 524; 9,71; 0; 0; 0,00; 0; 639; 655; 100; 59; 66 (Rico); 254; solto; solto; Neutro; 0; 9; 1,26; 8,99; 81; 1,17; 3,50; 5,80; 9;
1899; 0,00; 16; 768; 750; 521; 522; 10,31; 0; 0; 0,00; 0; 642; 596; 100; 59; 66 (Rico); 255; solto; solto; Neutro; 0; 9; 1,27; 8,99; 81; 1,17; 3,50; 5,80; 9;
1999; 0,00; 1; 757; 750; 521; 523; 10,28; 0; 0; 0,00; 0; 639; 660; 101; 59; 65 (Rico); 254; solto; solto; Neutro; 0; 9; 1,25; 8,99; 81; 1,17; 3,50; 5,80; 9;
2099; 0,00; 14; 762; 750; 521; 523; 11,21; 0; 0; 0,00; 0; 637; 598; 101; 59; 64 (Rico); 255; solto; solto; Neutro; 0; 9; 1,25; 8,99; 81; 1,17; 3,48; 5,80; 9;
2199; 0,00; 28; 772; 750; 521; 523; 12,19; 0; 0; 0,00; 0; 638; 657; 101; 59; 61 (Rico); 252; solto; solto; Neutro; 0; 9; 1,27; 8,99; 81; 1,17; 3,49; 5,80; 9;
2299; 0,00; -6; 732; 750; 521; 524; 13,14; 0; 0; 0,00; 0; 641; 599; 100; 59; 55 (Rico); 251; solto; solto; Neutro; 0; 9; 1,21; 8,99; 81; 1,17; 3,50; 5,81; 9;
2399; 0,00; -12; 747; 750; 521; 521; 14,04; 0; 0; 0,00; 0; 641; 667; 100; 59; 64 (Rico); 254; solto; solto; Neutro; 0; 9; 1,23; 8,99; 81; 1,17; 3,48; 5,80; 9;
2499; 0,00; -36; 708; 750; 521; 523; 13,89; 0; 0; 0,00; 0; 640; 633; 100; 59; 64 (Rico); 253; solto; solto; Neutro; 0; 9; 1,18; 8,99; 81; 1,17; 3,53; 5,81; 9;
2599; 0,00; -24; 735; 750; 521; 523; 13,84; 0; 0; 0,00; 0; 638; 637; 101; 59; 63 (Rico); 254; solto; solto; Neutro; 0; 9; 1,23; 8,99; 81; 1,17; 3,56; 5,80; 9;
2699; 0,00; -17; 741; 750; 522; 522; 14,21; 0; 0; 0,00; 0; 642; 664; 100; 59; 63 (Rico); 255; solto; solto; Neutro; 0; 9; 1,25; 8,99; 81; 1,17; 3,58; 5,80; 9;
2799; 0,00; -21; 741; 750; 522; 522; 14,11; 0; 0; 0,00; 0; 639; 631; 101; 59; 47 (Pobre); 241; solto; solto; Neutro; 0; 9; 1,25; 8,99; 81; 1,17; 3,58; 5,80; 9;
2899; 0,00; 7; 768; 750; 521; 522; 14,94; 0; 0; 0,00; 0; 640; 652; 101; 59; 62 (Rico); 252; solto; solto; Neutro; 0; 9; 1,30; 8,99; 81; 1,17; 3,58; 5,80; 9;
2999; 0,00; 6; 763; 750; 521; 523; 15,02; 0; 0; 0,00; 0; 639; 619; 101; 59; 54 (Rico); 246; solto; solto; Neutro; 0; 9; 1,28; 8,99; 81; 1,17; 3,56; 5,80; 9;
3099; 0,00; -10; 752; 750; 521; 523; 15,47; 0; 0; 0,00; 0; 638; 643; 101; 59; 55 (Rico); 252; solto; solto; Neutro; 0; 9; 1,26; 8,99; 81; 1,17; 3,55; 5,80; 9;
3199; 0,00; -20; 752; 750; 521; 523; 15,65; 0; 0; 0,00; 0; 640; 620; 101; 59; 47 (Pobre); 239; solto; solto; Neutro; 0; 9; 1,26; 8,99; 81; 1,17; 3,54; 5,81; 9;
3299; 0,00; 0; 732; 750; 521; 523; 15,98; 0; 0; 0,00; 0; 639; 656; 101; 59; 60 (Rico); 254; solto; solto; Neutro; 0; 9; 1,22; 8,99; 81; 1,17; 3,54; 5,80; 9;
3400; 0,00; 7; 754; 750; 521; 524; 16,17; 0; 0; 0,00; 0; 641; 620; 100; 59; 52 (Rico); 248; solto; solto; Neutro; 0; 9; 1,26; 8,99; 81; 1,17; 3,54; 5,80; 9;
3499; 0,00; 36; 744; 750; 521; 523; 16,03; 0; 0; 0,00; 0; 640; 668; 100; 59; 57 (Rico); 253; solto; solto; Neutro; 0; 9; 1,24; 8,99; 81; 1,17; 3,54; 5,80; 9;
3599; 0,00; 22; 757; 750; 521; 523; 16,57; 0; 0; 0,00; 0; 639; 609; 101; 59; 58 (Rico); 252; solto; solto; Neutro; 0; 9; 1,27; 8,99; 81; 1,17; 3,55; 5,80; 9;
3699; 0,00; 27; 768; 750; 521; 521; 17,30; 0; 0; 0,00; 0; 641; 648; 100; 59; 55 (Rico); 251; solto; solto; Neutro; 0; 9; 1,28; 8,99; 81; 1,17; 3,53; 5,80; 9;
3799; 0,00; 23; 764; 750; 521; 520; 17,62; 0; 0; 0,00; 0; 639; 609; 100; 59; 48 (Pobre); 240; solto; solto; Neutro; 0; 9; 1,27; 8,99; 81; 1,17; 3,52; 5,81; 9;
3899; 0,00; -30; 738; 750; 521; 523; 17,19; 0; 0; 0,00; 0; 638; 642; 101; 59; 52 (Rico); 251; solto; solto; Neutro; 0; 9; 1,22; 8,99; 81; 1,17; 3,51; 5,81; 9;
3999; 0,00; -1; 762; 750; 521; 523; 17,30; 0; 0; 0,00; 0; 638; 609; 101; 59; 46 (Pobre); 241; solto; solto; Neutro; 0; 9; 1,27; 8,99; 81; 1,17; 3,52; 5,81; 9;
4099; 0,00; 10; 762; 750; 521; 522; 18,88; 0; 0; 0,00; 0; 639; 645; 100; 59; 51 (Rico); 251; solto; solto; Neutro; 0; 9; 1,26; 8,99; 81; 1,17; 3,51; 5,81; 9;
4199; 0,00; -16; 736; 750; 521; 520; 18,74; 0; 0; 0,00; 0; 640; 610; 100; 59; 46 (Pobre); 239; solto; solto; Neutro; 0; 9; 1,24; 8,99; 81; 1,17; 3,56; 5,91; 9;
4299; 0,00; 23; 755; 750; 521; 523; 18,65; 0; 0; 0,00; 0; 640; 642; 100; 59; 54 (Rico); 254; solto; solto; Neutro; 0; 9; 1,25; 8,99; 81; 1,17; 3,51; 5,81; 9;
4399; 0,00; 9; 763; 750; 521; 523; 19,07; 0; 0; 0,00; 0; 639; 611; 101; 59; 44 (Pobre); 241; solto; solto; Neutro; 0; 9; 1,28; 8,99; 81; 1,17; 3,56; 5,86; 9;
4499; 0,00; 18; 762; 750; 521; 523; 19,13; 0; 0; 0,00; 0; 639; 644; 101; 59; 48 (Pobre); 251; solto; solto; Neutro; 0; 9; 1,26; 8,99; 81; 1,17; 3,51; 5,80; 9;
4599; 0,00; 7; 764; 750; 521; 521; 18,80; 0; 0; 0,00; 0; 641; 612; 100; 59; 41 (Pobre); 239; solto; solto; Neutro; 0; 9; 1,28; 8,99; 81; 1,17; 3,55; 5,86; 9;
4699; 0,00; -11; 757; 750; 521; 518; 19,85; 0; 0; 0,00; 0; 640; 638; 101; 59; 52 (Rico); 252; solto; solto; Neutro; 0; 9; 1,28; 8,99; 81; 1,17; 3,58; 5,91; 9;
4799; 0,00; 2; 766; 750; 520; 522; 20,97; 0; 0; 0,00; 0; 641; 614; 100; 59; 51 (Rico); 249; solto; solto; Neutro; 0; 9; 1,28; 8,99; 81; 1,17; 3,54; 5,86; 9;
4899; 0,00; 2; 740; 750; 521; 522; 21,22; 0; 0; 0,00; 0; 642; 644; 100; 59; 51 (Rico); 253; solto; solto; Neutro; 0; 9; 1,23; 8,99; 81; 1,17; 3,52; 5,80; 9;
4999; 0,00; 8; 770; 750; 520; 524; 21,75; 0; 0; 0,00; 0; 641; 611; 101; 59; 51 (Rico); 249; solto; solto; Neutro; 0; 9; 1,27; 8,99; 81; 1,17; 3,50; 5,80; 9;
5099; 0,00; 17; 779; 750; 520; 521; 21,93; 0; 0; 0,00; 0; 641; 622; 100; 59; 51 (Rico); 254; solto; solto; Neutro; 0; 9; 1,28; 8,99; 81; 1,17; 3,49; 5,80; 9;
5199; 0,00; 9; 745; 750; 520; 523; 21,96; 0; 0; 0,00; 0; 639; 611; 101; 59; 51 (Rico); 252; solto; solto; Neutro; 0; 9; 1,23; 8,99; 81; 1,17; 3,49; 5,80; 9;
5299; 0,00; -2; 753; 750; 520; 523; 21,76; 0; 0; 0,00; 0; 639; 627; 101; 59; 50 (Pobre); 250; solto; solto; Neutro; 0; 9; 1,24; 8,99; 81; 1,17; 3,48; 5,80; 9;
5399; 0,00; -14; 749; 750; 520; 522; 22,96; 0; 0; 0,00; 0; 641; 612; 100; 59; 50 (Pobre); 249; solto; solto; Neutro; 0; 9; 1,24; 8,99; 81; 1,17; 3,50; 5,81; 9

```

Fonte: Autor.

Figura 65 – Arquivo de *log* importado na ferramenta Excel.

ARQUIVO

PÁGINA INICIAL

INSERIR

LAYOUT DA PÁGINA

FÓRMULAS

DADOS

REVISÃO

EXIBIÇÃO

TEAM

Do Access

Da Web

De Texto

De Outras Fontes

Conexões Existentes

Atualizar Tudo

Conexões

Propriedades

Editar Links

Conexões

Classificar

Filtro

Limpar

Reaplicar

Avançado

Classificar e Filtrar

Texto para Colunas

Preenchimento Relâmpago

Remover Duplicatas de Dados

Validação

Consolidar

Teste de Hipóteses

Relações

Desagrupar Subtotal

Aggrupar

Desagrupar

Subtotal

Strutura de Tópicos

Pasta1 - Excel

Pereira, Bruno

A1

<

Fonte: Autor.

3.7.9 Automatização do ensaio de identificação

Um dos recursos mais importantes da aplicação de monitoramento consiste no controle automático do ensaio de identificação do modelo do sistema formado pelo veículo equipado com a ECU deste trabalho, modelo este utilizado para o projeto do controlador de cruzeiro. Especificamente neste projeto, o procedimento para a identificação do sistema é realizado com base na resposta ao degrau aplicado à entrada do sistema, que no caso corresponde à referência de torque. A saída, por sua vez, corresponde à velocidade do veículo, medida em km/h.

Ao clicar no botão destacado na [Figura 66](#), o usuário é capaz de definir todos os parâmetros do ensaio:

- Amplitude do degrau aplicado à referência de torque (0 a 100%);
- Instante da aplicação do degrau (em segundos);
- Tempo de medição, que equivale a duração do ensaio (em segundos);
- Taxa (período) de amostragem dos dados lidos da ECU (em milissegundos).

Figura 66 – Controle do ensaio de identificação pela aplicação de monitoramento.

Fonte: Autor.

Uma vez que o usuário define estes parâmetros e clica no botão “INICIAR”, o controle do ensaio de identificação segue o fluxograma ilustrado na [Figura 67](#), onde se destacam os seguintes estados:

1. **Inativo:** Estado inicial em que a aplicação obtém o controle do pedal de aceleração do veículo e aplica o valor de referência de torque mínimo, equivalente a condição de pedal totalmente solto. A partir deste momento, portanto, o controle do motor passa a ser da aplicação, restando ao condutor apenas o controle da direção¹⁴. Ao final deste estado, uma contagem de 5 segundos é disparada para início do ensaio.
2. **Iniciando:** Estado transitório que aguarda a contagem inicial dos 5 segundos, exibindo-a para usuário através da caixa de texto “*Status*”.
3. **Aguardando degrau:** Este estado dá início à aquisição dos dados da ECU que são utilizados na identificação da planta, obedecendo o período de amostragem definido pelo usuário. Além disto, após decorrido o tempo do instante do degrau, um comando é enviado à ECU para alterar o valor de sua referência de torque de acordo com a amplitude do degrau, parâmetro também definido pelo usuário¹⁵.
4. **Degrau aplicado:** Após a aplicação do degrau, este estado apenas aguarda pelo fim do experimento. Vale lembrar que neste estado a aquisição de dados da ECU ainda permanece ativa.
5. **Fim:** Neste estado todas as aquisições de dados da ECU são suspensas, sendo o controle do motor retornado ao condutor do veículo. Além disto, o usuário é avisado, através de uma mensagem, de que o teste foi realizado com sucesso, ficando a seu cargo salvar os dados em arquivo, através do botão “Salvar Dados”.
6. **Abortar:** Este estado é executado apenas quando se deseja abortar a tomada de dados, seja por decisão do condutor do veículo ou por decisão do usuário que está no comando da aplicação. No primeiro caso, o condutor pode abortar o teste pisando na embreagem, no freio ou simplesmente desengatando o veículo. Já no segundo caso, ao clicar no botão “ABORTAR” (destacado na [Figura 68](#)), o usuário também é capaz de cancelar a qualquer momento o teste em andamento. Uma vez executado, este estado suprime qualquer aquisição em andamento e exibe uma mensagem ao usuário notificando que o ensaio foi abortado.
7. **Falha de comunicação:** Este estado é executado quando houver falha de qualquer comando enviado à ECU, seja comando de aquisição de dados ou de controle do pedal de aceleração (ref. de torque). Uma vez executado, este estado suprime qualquer aquisição em andamento e exibe uma mensagem ao usuário notificando que o ensaio foi cancelado por falha de comunicação com a ECU.

¹⁴ Em caso de emergência, o condutor também poderá acionar os freios, embreagem ou ainda desengatar o veículo, com intuito de abortar o ensaio.

¹⁵ Este comando enviado à ECU corresponde ao mesmo que é enviado quando o usuário opta por controlar o motor por meio do controle deslizante da aplicação, conforme descrito na [subseção 3.7.4](#).

Figura 67 – Fluxograma do controle do ensaio de identificação.

Figura 68 – Ensaio de identificação em andamento na aplicação de monitoramento.

Fonte: Autor.

Ao término da tomada de dados, o usuário conta com uma ferramenta para auxiliá-lo na visualização gráfica dos dados obtidos no experimento. Ao clicar no botão “Ver Dados”, a aplicação traça um gráfico com os dados de entrada (ref. de torque) e saída (velocidade) do sistema, possibilitando ao usuário uma rápida inspeção visual, de modo que o mesmo possa avaliar a necessidade de se refazer o experimento. A título de exemplo, a [Figura 69](#) exhibe as curvas obtidas em um ensaio realizado neste trabalho com o veículo Polo.

Figura 69 – Visualização gráfica dos dados obtidos no ensaio de identificação.

Fonte: Autor.

Da mesma forma que ocorre na aquisição de dados de monitoramento, a aplicação disponibiliza uma opção para salvar em arquivo de texto os dados do ensaio realizado. O formato dos dados neste arquivo é o mesmo do arquivo de *log* de monitoramento, descrito na [subseção 3.7.8](#). Todavia, apenas parte dos parâmetros da ECU são lidos e disponibilizados em arquivo, como mostra a [Figura 70](#).

Figura 70 – Arquivo de *log* do ensaio de identificação importado na ferramenta Excel.

	A	B	C	D	E	F	G	H	I
1	Monitoramento e Diagnóstico - ECU v2.0								
2									
3	Data: 13/08/2016								
4	Hora: 16:06								
5									
6	Versão da aplicação: 2.7.2.0								
7	Firmware Gerenciamento: 1.6.1								
8	Firmware Sincronismo: 1.3.1								
9	Firmware Comunicação: 1.3.7								
10									
11	Ensaio para identificação do sistema 'veículo+ECUv2'								
12	Entrada u(t): Ref de Torque (pedal)								
13	Saída y(t): Velocidade (km/h)								
14	Taxa de amostragem (ms): 50								
15	Tempo de medição (s): 40								
16	Instante do degrau (s): 5								
17	Amplitude do degrau: 40								
18	Modo de operação: Normal								
19	Marcha Engatada: 3								
20	Temperatura do Motor: 88,48°C								
21									
22	t0_Aquisicao(ms)	tf_Aquisicao(ms)	RefTorque	Veloc(km/h)	RefTorque (ECU)	VelocCAN (km/h)	Rotacao		
23	13	16	0	15,04	0	14	713		
24	63	80	0	15,09	0	14	713		
25	113	128	0	15,21	0	14	733		
26	163	176	0	15,09	0	14	733		
27	213	224	0	15,01	0	13	713		
28	263	273	0	14,97	0	13	730		
29	313	320	0	14,91	0	13	730		
30	363	368	0	14,93	0	13	714		
31	413	416	0	14,96	0	14	714		

Fonte: Autor.

Ressalta-se que, apesar da presença de mecanismos de segurança para abortar o ensaio¹⁶, todos os testes foram realizados com auxílio de um dinamômetro inercial, sendo o veículo devidamente amarrado a ele. Além disto, vale lembrar que, se por algum motivo a aplicação de monitoramento falhar durante o ensaio, a embreagem poderá ser utilizada como uma segunda opção para abortar o experimento em andamento, desarmando, neste caso, o controle do pedal de aceleração pela aplicação, conforme descrito na [subseção 3.7.4](#). Como medida extrema para abortar o experimento, o condutor do veículo conta ainda com um botão de emergência que pode ser acionado a qualquer momento para desligar de imediato todos os atuadores do motor, conforme será abordado na [subseção 3.8.12](#).

¹⁶ Conforme mencionado, o ensaio de identificação pode ser abortado a qualquer momento mediante acionamento do freio ou da embreagem, desengatando o veículo ou ainda clicando no botão “ABORTAR” da aplicação.

3.8 Outros Aperfeiçoamentos

As seções a seguir abordam outros aperfeiçoamentos que foram realizados no *firmware* da ECU v2.0. Vale ressaltar que a boa parte destes recursos não estavam presentes no *firmware* desenvolvido em 2013.

3.8.1 Controle de partida

No *firmware* da ECU de 2013, o controle de partida consistia em se manter a borboleta na abertura determinada apenas pela sua mola de retorno, posição esta denominada *Limp Home* (LH)¹⁷. Desta forma, a borboleta eletrônica permanecia com esta abertura até que a rotação do motor ultrapassasse 800 RPM, para então liberar o controle de rotação para atuar no regime de marcha lenta.

Apesar de funcionar bem, esta estratégia fornecia um giro muito alto para a partida, atingindo, em algumas ocasiões, em torno de 2000 RPM. Assim, o controle de partida atual foi concebido com intuito de fornecer uma partida mais suave do motor, além de melhorar a sua partida em baixas temperaturas.

A estratégia atual é bem parecida com a de 2013, porém agora a posição de partida da borboleta é variável de acordo com a temperatura do motor, como mostra o gráfico da [Figura 71](#). Além disto, é importante notar que esta posição de partida é menor do que a posição de *Limp Home* (que equivale a uma abertura em torno de 6,7%), o que garante uma partida mais suave, principalmente na condição de motor quente.

¹⁷ A borboleta eletrônica assume esta posição quando a tensão aplicada ao seu servomecanismo é nula, situação típica de uma falha no mecanismo. Com esta abertura da borboleta de admissão, o motor é capaz de fornecer aproximadamente 2000 RPM de giro, condição suficiente para que o condutor consiga conduzir o veículo até um local de reparo.

Figura 71 – Posição da borboleta para a partida do motor.

Fonte: Autor.

Outro recurso do controle de partida atual consiste em um atraso de liberação do controle de marcha lenta após a rotação do motor superar a referência de 800 RPM, de maneira a evitar qualquer instabilidade na rotação do motor após a sua partida, garantindo assim uma transição suave para o regime de marcha lenta. Da mesma forma que a posição da borboleta, este atraso também é dependente da temperatura do motor, conforme descreve o gráfico da [Figura 72](#).

Figura 72 – Atraso do controle de partida.

Fonte: Autor.

Menciona-se ainda o ganho de injeção para partida com motor frio, com intuito de enriquecer a mistura a/c para melhorar o desempenho do motor nesta condição.

Analogamente aos parâmetros anteriores, este ganho também é dependente da temperatura do motor, como mostra a curva da [Figura 73](#).

Figura 73 – Ganho de injeção para partida a frio.

Fonte: Autor.

É importante ressaltar que o controle de partida também é responsável pela pressurização do combustível através do acionamento do relé da bomba de combustível por 1 segundo antes do acionamento do motor de partida.

3.8.2 Estratégia para motor frio

Testes práticos em 2013 mostraram que para se melhorar o desempenho do motor quando a sua temperatura está abaixo do ponto de operação é necessário enriquecer a mistura a/c. Isto era feito aplicando um ganho no tempo de injeção de até 20%. No entanto, no trabalho atual foi verificado que este ganho pode ser reduzido para 10%, de acordo com a curva da [Figura 74](#). É importante notar também que este ganho não é necessário quando a temperatura excede 50°C, sendo que a partir deste ponto a ECU passa adotar estratégia para motor quente.

Figura 74 – Ganho de injeção para motor frio.

Fonte: Autor.

Além do ganho de injeção, outro comportamento da ECU para motor frio consiste em se limitar as variações na válvula borboleta, evitando assim pequenos afogamentos do motor, além de melhorar a dirigibilidade do veículo nesta condição. Com isto, o condutor do veículo não precisa se preocupar em pressionar suavemente o pedal do acelerador, pois neste caso a ECU já seria responsável por suavizar a abertura da borboleta eletrônica.

Especificamente no projeto atual, para abrir a borboleta o *firmware* implementa um filtro que limita a sua variação máxima de posição em aproximadamente 20% por segundo. Como resultado deste filtro, a borboleta eletrônica levaria em torno de 5 segundos para a sua abertura completa, caso o pedal de aceleração fosse pressionado bruscamente. Vale lembrar que o filtro é mais ameno para o fechamento da borboleta, de modo a evitar que o veículo continue acelerando mesmo após o motorista liberar o pedal depois de uma aceleração acentuada.

3.8.3 Fator de correção de acordo com a tensão da bateria

O controle do tempo de injeção e do tempo de carregamento da bobina de ignição devem levar em conta o valor da tensão da bateria do veículo. No caso da ignição, por exemplo, a ECU deverá aumentar o tempo de carregamento sempre que houver uma queda na tensão da bateria, de tal modo que a centelha de combustão forneça a mesma energia que no caso em que a bateria está com sua tensão normal de operação, em torno de 14V. O mesmo deve ocorrer para injeção, ou seja, o tempo de injeção deve ser maior com a queda da tensão de bateria para que a quantidade de combustível injetada não sofra variação.

Para contornar este problema, o *firmware* atual do μC de Gerenciamento aplica um fator de correção multiplicativo no tempo de injeção e no tempo de carregamento

da bobina de ignição, fator este que é função da tensão de bateria. Para determinação deste fator de correção, diversas medições foram realizadas com a ECU original do veículo, e os dados resultantes foram interpolados por uma função exponencial, como mostra a Figura 75.

Figura 75 – Fator de correção dos tempos de injeção e ignição em função da tensão de bateria.

Fonte: Autor.

Os trabalhos práticos demonstraram que quedas na tensão da bateria ocorrem principalmente durante a partida do motor e quando partes elétricas do veículo (como, por exemplo, os faróis e os desembaçadores de vidro) são acionadas durante o regime de marcha lenta com a rotação em torno ou abaixo de 800 RPM. Nestes casos, a tensão da bateria tende a cair abaixo de 14V, o que explica o fator de correção ser maior do que 1 nesta condição, como mostra a curva da Figura 75. Vale lembrar que acima de 14V a ECU não aplica nenhuma correção nos tempos mencionados, o que equivale utilizar fator de correção igual a 1.

3.8.4 Cálculo da velocidade do veículo

Apesar da velocidade do veículo estar disponível para leitura no barramento CAN, o valor fornecido não possui precisão suficiente para o funcionamento adequado do controle de cruzeiro, visto que neste caso a informação da velocidade é disponibilizada apenas em 1 *byte* de informação na mensagem CAN e possui fator de conversão de 1,333. Em outras palavras, utilizando o valor disponível na CAN, a menor variação de velocidade possível de ser lida pela ECU corresponde a 1,333 km/h, o que inviabiliza o seu uso para realimentação do controlador de cruzeiro.

Para resolver esta situação, a ECU atual realiza a leitura do sinal proveniente do sensor de velocidade, que corresponde a um sensor de efeito Hall que, acoplado ao câmbio do veículo, fornece uma onda quadrada com frequência equivalente à velocidade do veículo, na proporção 1 Hz para 1 km/h¹⁸. Analogamente à leitura do sinal de rotação, a leitura deste sinal de velocidade é realizada pelo microcontrolador de Gerenciamento por meio de interrupção externa de alta prioridade, de modo a evitar erros no cálculo da velocidade final. Além disto, a medição do período do sinal é realizada com auxílio de um *timer* de alta resolução, de modo a se alcançar uma boa precisão nesta medida.

Os primeiros testes revelaram a presença de ruídos formados por pulsos de curta duração no sinal medido, fato que ocasionou erro no cálculo da velocidade. Para contornar este problema, o *firmware* foi preparado para ignorar pulsos com duração inferior a 1 milissegundo, solucionando o problema em questão.

3.8.5 Identificação de marcha

A identificação de marcha é necessária para detectar situações em que o veículo se encontra engatado, de tal forma que a ECU possa adotar diferentes comportamentos nestas situações. No *firmware* atual da ECU, a condição de marcha desengatada é utilizada para desarmar o controle de cruzeiro, enquanto a condição de marcha engatada é utilizada para ativação do filtro da borboleta. Além disto, o valor da marcha é utilizado pela aplicação de monitoramento para abortar o ensaio de identificação na condição de veículo desengatado (marcha engatada igual a zero), além de ser utilizado também para exibição em tela e para gravação em *log* durante o monitoramento da ECU.

Para identificar a marcha engatada, a ECU se baseia na relação entre a rotação do motor e a velocidade do veículo, relação esta que pode ser obtida através de experimentos práticos. A Figura 76, por exemplo, mostra a aquisição realizada em 3ª marcha, em que se observa que a relação para esta marcha pode ser aproximada por uma reta. As demais aquisições realizadas com as outras marchas também forneceram relações lineares, porém cada aquisição forneceu uma reta com um determinado coeficiente angular característico da marcha utilizada. Desta forma, este coeficiente angular pode ser utilizado pelo *firmware* para identificação da marcha.

¹⁸ Um sinal com frequência de 60 Hz, por exemplo, indicaria uma velocidade de 60 km/h.

Figura 76 – Relação rotação/velocidade da 3ª marcha.

Fonte: Autor.

Vale lembrar que o algoritmo da ECU que realiza a detecção da marcha deve ser flexível para levar em conta dispersões dos valores medidos, ou seja, as medições realizadas pelo *firmware* podem fornecer relações que estejam ligeiramente fora da reta definida pela interpolação dos dados da aquisição. Por esta razão, o algoritmo implementado em *firmware* analisa a distância da relação medida para a reta respectiva de cada marcha, sendo que a marcha final será assumida como aquela em que esta distância é mínima. Além disto, o algoritmo também considera uma tolerância máxima, de tal forma que a distância mínima obtida esteja dentro deste valor de tolerância. Caso contrário, o veículo é considerado como desengatado.

3.8.6 *Debounce* de sensores digitais

O *debounce* de sinais digitais é necessário para se filtrar ruídos que possam prejudicar a leitura destes sinais. Em geral, estes ruídos são formados por pulsos de curta duração que podem ser gerados a partir de chaves mecânicas, fontes externas ou até mesmo de outros sinais presentes na placa.

A Figura 77 ilustra o funcionamento de um filtro *debounce*. Nela é possível observar que o sinal filtrado, referenciado na figura por “*Debounced*”, apenas muda de estado quando o sinal original permanece estável por um certo intervalo de tempo conhecido por tempo de *debounce*. No caso do exemplo ilustrado, este tempo corresponde a 20 ms, o que significa que pulsos com duração inferior a 20 ms são filtrados.

Figura 77 – Ilustração do processo de *debounce* para filtragem de sinais digitais.

Fonte: Donzellini e Ponta (2016).

É importante notar que o filtro de *debounce* insere um atraso no sinal original, da mesma forma que ocorre quando se utiliza um filtro para sinal analógico. No exemplo da Figura 77, o sinal filtrado apenas muda de estado 20 ms após a mudança do sinal original, assumindo que este permanece estável após a sua última transição. Portanto, é importante que este atraso não seja prejudicial para a aplicação em questão.

No caso da ECU v2.0, a leitura dos sensores digitais ocorre a cada 10 ms, sendo que o *firmware* retém as últimas 4 amostras para análise, o que resulta em um tempo de *debounce* de 40 ms. Os sensores filtrados por este método são:

1. Sinal da linha 15 (chave da ignição);
2. Sinal de acionamento do controle de cruzeiro, proveniente de um botão instalado no painel do motorista;
3. Sinal dos freios;
4. Sinal da embreagem;
5. Sinal de falha da ponte-H presente no *hardware* da ECU v2.0;
6. Sinal de acionamento do botão de emergência.

Vale enfatizar que os sinais de rotação e de velocidade não podem ser filtrados por este método, pois são sinais de alta frequência que necessitam ser lidos e processados com o menor atraso possível.

3.8.7 Filtro para aceleração e desaceleração com veículo engatado

Quando o veículo está engatado, o *firmware* do bloco de Gerenciamento emprega um filtro para limitar a velocidade de abertura da válvula borboleta eletrônica, com objetivo de eliminar solavancos no veículo quando o pedal de aceleração é pressionado bruscamente, o que resulta em maior conforto aos usuários do veículo.

No projeto de 2013, este filtro atuava diretamente no sinal de entrada da ECU, que no caso correspondia à referência de rotação, dada pelo pedal de aceleração (PEREIRA, 2013). Embora o resultado tenha sido satisfatório, o projeto atual optou por filtrar diretamente a referência de posição da borboleta, de modo a reduzir o atraso inserido pelo filtro na aceleração do veículo.

Especificamente a cada 10 ms, o μC de Gerenciamento calcula a diferença entre a referência de posição da borboleta determinada pelo *firmware* naquele instante e a referência determinada no laço anterior, para então verificar se esta variação calculada é maior do que um certo valor limite. Em caso positivo, a referência da borboleta é alterada para que a sua variação não extrapole o limite imposto.

Vale mencionar que este limite, determinado empiricamente por meio de testes no dinamômetro, depende da direção da variação de posição da borboleta (abertura ou fechamento) e também do valor da referência atual. No caso de uma abertura, por exemplo, o limite assume dois valores, como mostra a equação 3.6:

$$Limite = \begin{cases} \frac{0,16\%}{10ms} = 16\%/s, & \text{se } Ref < 10\% \\ \frac{1,12\%}{10ms} = 112\%/s, & \text{se } Ref \geq 10\% \end{cases} \quad (3.6)$$

Pela equação anterior, é possível observar que a primeira abertura da borboleta, quando a sua posição é inferior a 10%, é realizada com um filtro mais acentuado, de modo que o resultado seja uma abertura mais lenta. Em seguida, ao atingir aproximadamente 10% de abertura, o filtro é amenizado, liberando assim uma abertura mais rápida da borboleta eletrônica.

3.8.8 Elevação da marcha lenta para arranque do veículo

A estratégia de se elevar a rotação de marcha lenta quando a embreagem é acionada foi adotada com intuito de melhorar a arrancada do veículo. A Figura 78 ilustra esta situação, em que se observa a seguinte sequência de fatos:

1. O condutor aciona a embreagem por volta de 2,5 segundos, instante em que a ECU eleva a referência de rotação de marcha lenta para 1000 RPM;
2. O ponto da embreagem se dá entre 6 e 8 segundos, intervalo em que o controle de marcha lenta recebe a carga do veículo engatado, fato que, por sua vez, se evidencia pela queda da rotação do motor;
3. O veículo sai do estado de repouso por volta do instante 7,5 segundos, quando então começa a ganhar velocidade devido à atuação do controle de marcha lenta no sentido de corrigir a rotação do motor;
4. O condutor solta o pedal da embreagem em torno do instante 9 segundos, quando então o controle de marcha lenta passa a ser responsável por manter o veículo em movimento a uma velocidade em torno de 8,5 km/h, com primeira marcha engatada.

Figura 78 – Elevação da rotação de marcha lenta para 1000 RPM quando a embreagem é acionada.

Fonte: Autor.

É importante notar que neste exemplo o pedal de aceleração não foi utilizado pelo condutor para colocar o veículo em movimento, fato que se traduz na referência de torque se manter nula durante todo o teste.

3.8.9 Aumento da troca de dados no barramento CAN

Através do bloco de Comunicação, a ECU de 2013 utilizava a rede CAN apenas para ler a velocidade do veículo e enviar o valor da rotação do motor, neste último caso com intuito de alimentar o tacômetro do painel do veículo. Já na ECU atual, o *firmware* do μ C de Comunicação foi modificado para enviar, além da rotação, outros parâmetros do motor ao barramento CAN do veículo, tais como:

- Temperatura do motor;
- Acionamento do freio;
- Acionamento da embreagem;
- Acionamento do pedal de aceleração;
- Referência de torque (valor do pedal de aceleração);
- Comando para ligar a luz de EPC (falha da ECU) no painel do veículo, de acordo com as variáveis de erro dos blocos de Gerenciamento e de Sincronismo;
- Comando para ligar o ar condicionado;
- Sinal de consumo, para alimentar o *display* do painel que mostra o consumo instantâneo em km/L.

Com relação à recepção, além da velocidade, o *firmware* atual está preparado para receber os seguintes parâmetros do barramento CAN:

- Nível de combustível, juntamente com a indicação de entrada na reserva;
- Acionamento do freio de estacionamento;
- Estado da luz de bateria presente no painel do veículo;
- Temperatura ambiente;
- Estado de luzes externas (setas, pisca alerta, lanterna e luz de ré);
- Estado das portas do veículo, incluindo o capô e o porta malas;
- Valor do hodômetro;
- Estado do acionamento do ar condicionado;
- Requisições OBD-II.

Vale lembrar que a maior parte destes parâmetros lidos da CAN tem como finalidade apenas realimentar a aplicação de monitoramento da ECU para prover ao usuário um computador de bordo alternativo. Já o envio de parâmetros do motor tem como finalidade realimentar outros módulos presentes no veículo, de tal forma que os possibilitem identificar o regime do motor.

Além disto, é importante notar que neste veículo o comando do ar condicionado é realizado pela ECU do motor, ou seja, quando o usuário aciona o botão no painel, o ar condicionado apenas será acionado mediante envio de comando específico no barramento CAN pela ECU do motor.

3.8.10 Suporte para diagnóstico via OBD-II

O *firmware* do bloco de Comunicação da ECU atual foi aperfeiçoado para possibilitar o diagnóstico da ECU v2.0 através de um leitor OBD-II, como mostram as Figuras 79 e 80. Dentre os parâmetros que podem ser lidos da ECU atual por meio deste protocolo, destacam-se:

- Estado de falha da ECU (luz de EPC do painel);
- Temperatura do motor;
- Posição da borboleta (sensor TPS);
- Rotação do motor;
- Velocidade do veículo;
- Avanço de ignição;
- Temperatura do ar admitido;
- Pressão do ar admitido (sensor MAP);
- Tensão do sensor lambda;
- Tempo de funcionamento do motor;
- Nível de combustível;
- Tensão da bateria;
- Temperatura ambiente;
- Valor do pedal de aceleração;
- Sinal de atuação da borboleta (PWM);

- Fluxo instantâneo de combustível (em L/h);
- Referência de torque.

Figura 79 – Adaptador OBD-II conectado à ECU v2.0.

Fonte: Autor.

Figura 80 – Leitura de parâmetros da ECU v2.0 por meio do aplicativo *Torque*.

Fonte: Autor.

O código embarcado no μC de Comunicação para responder às requisições do protocolo OBD-II foi desenvolvido com auxílio de [Pereira \(2015\)](#).

3.8.11 Detecção de falhas de hardware

Desde o projeto inicial de 2013, o *firmware* da ECU foi desenvolvido com capacidade para se detectar erros de *hardware* decorrentes de mal funcionamento de sensores e atuadores, rompimento de cabos, e em alguns casos até danos em circuitos condicionadores da placa. Assim, ao se detectar um erro de *hardware*, a estratégia de 2013 consistia em se desligar os atuadores do motor, de maneira que o usuário fosse capaz de resolver o problema para então religar a ECU e permitir nova partida do motor.

Embora a detecção seja eficaz, a ação tomada pela ECU de desligar o motor quando ocorrer erro de *hardware* pode ser altamente prejudicial para a segurança do veículo. Caso uma falha desta natureza ocorresse, por exemplo, em uma situação de ultrapassagem, o motor seria desligado e a demanda por aceleração exigida pelo condutor não seria atendida, o que poderia resultar em um acidente.

Desta forma, para não prejudicar a segurança do veículo, a ECU atual recebeu melhorias para manter, sempre que possível, o motor funcionando, mesmo em condições de falha de *hardware*. Para isto, o *firmware* do bloco de Gerenciamento toma algumas ações de acordo com a falha ocorrida, como mostra a [Tabela 1](#):

Tabela 1 – Relação das falhas de *hardware* detectáveis pela ECU v2.0.

Código ¹	Descrição	Ação tomada pelo <i>firmware</i>
0x00	Sem erro	Funcionamento normal
0x01	Falha no servomecanismo da válvula borboleta eletrônica (VB)	Desliga o controle da VB para forçar a sua posição em <i>Limp Home</i> . ²
0x02	Falha no sensor Pedal 1	Assume pedal pouco pressionado (10%) para que o motor forneça aproximadamente 2000 RPM.
0x03	Falha no sensor Pedal 2	Apenas seta <i>flag</i> de erro (liga EPC). ³
0x04	Falha no sensor TPS 1	Apenas seta <i>flag</i> de erro (liga EPC). ³
0x05	Falha no sensor TPS 2	Assume TPS na posição de <i>Limp Home</i> e desliga o controle da VB para forçar a sua posição em <i>Limp Home</i> . ²
0x06	Falha no sensor MAP	Assume pressão de 60 kPa até a partida do motor, e então assume 40 kPa constante para manter o motor funcionando.
0x07	Falha no sensor de temperatura do ar	Assume temperatura do ar igual a 30°C.
0x08	Falha no sensor de temperatura da água	Assume temperatura do motor igual a 80°C.
0x09	Falha na leitura da tensão de bateria	Assume bateria a 14V para evitar compensação dos atuadores.
0x0D	Falha no CI 33186 (Ponte-H da VB)	Desliga o controle da VB para forçar a sua posição em <i>Limp Home</i> . ²
0x0E	Falha no CI 33810 (<i>driver</i> de injeção e ignição)	Desliga os atuadores e evita a partida do motor. ⁴
0x0F	Falha no Injetor 1	Seta <i>flag</i> de erro (liga EPC) e mantém o motor funcionando com um cilindro a menos.
0x10	Falha no Injetor 2	Seta <i>flag</i> de erro (liga EPC) e mantém o motor funcionando com um cilindro a menos.
0x11	Falha no Injetor 3	Seta <i>flag</i> de erro (liga EPC) e mantém o motor funcionando com um cilindro a menos.
0x12	Falha no Injetor 4	Seta <i>flag</i> de erro (liga EPC) e mantém o motor funcionando com um cilindro a menos.

Além das ações mencionadas na tabela anterior, o *firmware* também adota as seguintes medidas complementares em caso de erro de *hardware*:

- ¹ O código de erro de *hardware* pode ser lido pela aplicação de monitoramento, descrita na [seção 3.7](#).
- ² Uma vez desligado, o controle da borboleta eletrônica somente é reativado após o reset do *firmware*, procedimento que pode ser realizado desligando e religando a linha 15 (chave da ignição).
- ³ Apesar de estarem conectados ao *hardware*, os sensores Pedal 2 e TPS 1 não estão sendo utilizados pelo *firmware* da ECU v2.0.
- ⁴ A detecção de erro do *driver* 33810 é realizada somente uma vez após o início do *firmware*, antes da partida do motor. Desta forma, o desligamento dos atuadores é realizado apenas com intuito de suprimir a partida do motor.

1. Desativação do controle de marcha lenta por avanço de ignição e redução dos ganhos do controle de marcha lenta por atuação na borboleta;
2. Desativação do controle lambda (controle da mistura a/c em malha fechada);
3. Desativação do aquecimento da sonda lambda;
4. Desativação do controle de cruzeiro (se ativo);
5. Alteração da rotação de marcha lenta para 1000 RPM;
6. Desativação do corte de injeção em desacelerações;
7. Acionamento do LED de erro de *hardware* para piscar a 5 Hz;
8. Envia comando no barramento CAN para acionamento da luz de EPC no painel.

3.8.12 Botão de emergência

Localizado próximo ao motorista, o botão de emergência (Figura 81) foi adicionado ao projeto com objetivo de melhorar a segurança dos trabalhos práticos de desenvolvimento da ECU v2.0. Conectado ao bloco de Gerenciamento, este botão, quando acionado, provoca o desligamento imediato dos atuadores do motor, sem a necessidade de se desligar a chave da ignição (linha 15). Com isto, é possível atuar rapidamente em uma situação de emergência, desligando o motor de forma rápida e segura para manter a sua integridade e a segurança das pessoas envolvidas no trabalho.

Figura 81 – Botão de emergência conectado à ECU v2.0.

Fonte: Autor.

De forma mais específica, o botão de emergência provoca o desligamento da injeção e da ignição, além de desligar também o sinal de atuação da borboleta eletrônica, com intuito de mantê-la fechada na posição de *Limp Home*. Para executar esta tarefa, o *firmware* do bloco de Gerenciamento, ao detectar o pressionamento do botão, desabilita as saídas do *driver* de injeção e ignição (CI 33810) e desliga também a ponte H (CI 33186) que alimenta o servomecanismo da borboleta eletrônica. Em seguida, a ECU aguarda o desligamento da chave de ignição (linha 15) para então resetar o sistema, permitindo assim uma nova partida do motor.

3.9 Resultados

A seções seguintes abordam os resultados obtidos por meio das principais malhas de controle inseridas ou aprimoradas no *firmware* da ECU deste trabalho.

3.9.1 Controle de torque em malha aberta - Dirigibilidade

Como mencionando anteriormente, o controle de torque da ECU deste trabalho foi implementado em malha aberta, de maneira a simplificar a solução final. Sendo assim, a validação deste controle se deu por meio de testes realizados com o veículo em dinamômetro inercial e também em pista, possibilitando, deste modo, uma análise qualitativa da dirigibilidade do veículo.

Os resultados obtidos através de testes práticos demonstraram que, apesar de simples, este controle de torque em malha aberta contribuiu para uma melhora significativa da dirigibilidade do veículo, quando comparado ao controle de rotação introduzido em 2013.

3.9.2 Curvas de Potência

No teste de potência, realizado com o veículo devidamente fixado ao dinamômetro inercial, o pedal de aceleração é totalmente pressionado para que o motor, por meio do controle provido pela ECU, entregue a máxima potência e o máximo torque. No presente trabalho, este teste foi realizado em 3ª marcha, com início em 1500 RPM e término em 5500 RPM. Além disto, o dinamômetro inercial é responsável por controlar a carga imposta ao veículo para garantir que o ensaio dure 20 segundos.

Testes de potência foram realizados com a ECU v2.0 deste trabalho e com a ECU original do veículo, com objetivo de permitir uma comparação de desempenho entre ambas. Pelas curvas da [Figura 82](#), nota-se que, de modo geral, a ECU deste projeto apresenta desempenho similar à ECU original do veículo, dado que as curvas são bastante similares. Os pontos máximos de cada curva foram apresentados na [Tabela 2](#).

Figura 82 – Curvas de potência e torque (ECU original x ECU v2.0).

Fonte: Autor.

Tabela 2 – Potência e torque máximos (ECU original x ECU v2.0).

Unidade Eletrônica	Potência máxima	Torque máximo
ECU original	104,1 cv a 5085 RPM	16,6 mkgf a 3600 RPM
ECU v2.0	105,6 cv a 4821 RPM	17,0 mkgf a 4194 RPM

A [Figura 83](#), por sua vez, exibe o resultado do ensaio realizado com os modos de operação normal e econômico¹⁹ da ECU deste trabalho. Os pontos máximos de cada curva deste ensaio foram destacados na [Tabela 3](#).

¹⁹ Lembrando que o modo econômico da ECU deste trabalho restringe a abertura máxima da borboleta eletrônica em 50% quando o pedal de aceleração é totalmente pressionado.

Figura 83 – Curvas de potência e torque (modo normal x modo econômico).

Fonte: Autor.

Tabela 3 – Potência e torque máximos (modo normal x modo econômico).

Modo de operação	Potência máxima	Torque máximo
Normal	105,6 cv a 4821 RPM	17,0 mkgf a 4194 RPM
Econômico	97,8 cv a 5085 RPM	16,2 mkgf a 3600 RPM

3.9.3 Controle da rotação de marcha lenta

Conforme mencionado na [seção 3.3](#), a primeira versão do controle da rotação de marcha lenta da ECU deste projeto era composta por um controlador PI com atuação exclusiva no ângulo de abertura da borboleta eletrônica, como ilustra o diagrama da [Figura 37](#). Entretanto, para aprimorar a resposta da marcha lenta, adicionou-se outra malha de controle composta por um controlador PD com atuação no avanço de ignição. Como resultado, a versão final do controle de marcha lenta da ECU atual contempla duas malhas de controle que, atuando simultaneamente na posição da borboleta eletrônica e no avanço de ignição (conforme diagrama da [Figura 41](#)), são responsáveis por manter a rotação do motor o mais próximo possível do seu valor de referência durante este regime.

Desta maneira, os testes para validação do controle de marcha lenta foram divididos em dois conjuntos, sendo o primeiro conjunto de testes executado com a primeira versão do controle e o segundo, com a sua versão final.

Além disto, os testes realizados tem como intuito analisar a eficácia do controle de marcha lenta, inclusive na presença de distúrbios. O controle lambda, por exemplo, pode ser visto como um pequeno distúrbio para o controle de marcha lenta, ao passo que pode causar pequenas oscilações na rotação de marcha lenta ao corrigir constantemente o tempo de injeção.

3.9.3.1 Primeira versão

O resultado da primeira versão do controle de marcha lenta exibido na [Figura 84](#) demonstra que é possível se controlar a rotação do regime de marcha lenta apenas por meio da borboleta eletrônica.

Figura 84 – Controle da rotação de marcha lenta a 800 RPM (versão sem atuação na ignição).

Fonte: Autor.

Entretanto, o regime de marcha lenta não se estabiliza com a referência de rotação em 700 RPM, conforme evidenciado na [Figura 85](#). Ressalta-se que este fato, entretanto,

não é um problema, dado que este valor de rotação não é habitual para o regime de marcha lenta do veículo em questão, de acordo com testes executados com a ECU original do veículo.

Figura 85 – Controle da rotação de marcha lenta a 700 RPM (versão sem atuação na ignição).

Fonte: Autor.

As Figuras 86 e 87 exibem, respectivamente, os testes realizados com o acionamento do ar condicionado e da carga elétrica do veículo durante o regime de marcha lenta. A menos de um queda na rotação do motor no instante em que o ar condicionado e a carga elétrica são acionados, nota-se que o regime de marcha lenta permaneceu estável durante estes testes. Vale lembrar também que os respectivos acionamentos ocorreram em torno de 5 segundos, sendo desligados em torno de 40 a 45 segundos. Por fim, destaca-se a elevação da rotação de marcha lenta para 900 RPM quando o ar condicionado é ligado, estratégia adotada pela ECU deste trabalho para conservar a carga elétrica do alternador do veículo.

Figura 86 – Controle da rotação de marcha lenta durante acionamento do ar condicionado (versão sem atuação na ignição).

Fonte: Autor.

Figura 87 – Controle da rotação de marcha lenta durante acionamento da carga elétrica do veículo (versão sem atuação na ignição).

Fonte: Autor.

A Figura 88 exibe a atuação do controle de marcha lenta quando a sua referência de rotação varia lentamente. Neste caso, é possível observar que a rotação do motor consegue acompanhar, na média, as variações impostas na referência. Entretanto, no teste seguinte em que a variação da referência é mais acentuada (Figura 89), a rotação somente a segue com um atraso considerável, fato que é esperado dado que a variação de posição da borboleta responde lentamente na rotação de marcha lenta, conforme discutido na subseção 2.1.4. Adianta-se que a versão final do controle de marcha lenta, que utiliza o avanço de ignição em conjunto com a borboleta eletrônica, tende a corrigir este problema.

Figura 88 – Controle da rotação de marcha lenta com variação lenta na referência (versão sem atuação na ignição).

Fonte: Autor.

Figura 89 – Controle da rotação de marcha lenta com variação rápida na referência (versão sem atuação na ignição).

Fonte: Autor.

3.9.3.2 Versão final

A [Figura 90](#) exibe o resultado da versão final do controle de marcha lenta deste trabalho para a referência definida em 800 RPM. Nela, é possível observar que o erro de rotação fica limitado a um valor máximo de 20 RPM, o que é um resultado excelente para uma rotação em torno de 800 RPM.

Figura 90 – Controle da rotação de marcha lenta a 800 RPM.

Fonte: Autor.

Através da versão final do controle de marcha lenta, foi possível alcançar a estabilidade deste regime mesmo quando a referência é definida em 700 RPM, conforme mostra a [Figura 91](#). Além disto, similarmente ao resultado anterior para referência de 800 RPM ([Figura 90](#)), nota-se que o erro se mantém limitado a um valor pequeno, em torno de 25 RPM.

Figura 91 – Controle da rotação de marcha lenta a 700 RPM.

Fonte: Autor.

Pelos resultados obtidos nas Figuras 92 e 93, os quais abordam, respectivamente, os testes executados com o acionamento do ar condicionado e da carga elétrica durante o regime de marcha lenta, é possível concluir que o uso do avanço de ignição é fundamental para corrigir efeitos transitórios decorrentes da aplicação de distúrbios no controle da rotação de marcha lenta. Comparando-se os resultados das Figuras 86 e 92, por exemplo, nota-se que no último caso, que fez uso do avanço de ignição, houve redução considerável do tempo para minimizar o erro de rotação introduzido pela carga do ar condicionado. Já quando se compara as situações das Figuras 87 e 93, fica claro que o uso do avanço de ignição contribui para manter o erro de rotação limitado a um valor pequeno durante regime de marcha lenta, mesmo na presença do distúrbio da carga elétrica do veículo.

Figura 92 – Controle da rotação de marcha lenta durante acionamento do ar condicionado.

Fonte: Autor.

Figura 93 – Controle da rotação de marcha lenta durante acionamento da carga elétrica do veículo.

Fonte: Autor.

Por fim, as Figuras 94 e 95 mostram os resultados obtidos quando se aplicam variações na referência de rotação do regime de marcha lenta. Diferentemente da primeira versão, observa-se que agora a rotação segue fielmente a sua referência, até mesmo no caso em que a variação é mais rápida (Figura 95). Vale enfatizar que ao final destes testes, a referência foi mantida em 600 RPM, situação destacada na Figura 96.

Figura 94 – Controle da rotação de marcha lenta com variação lenta na referência.

Fonte: Autor.

Figura 95 – Controle da rotação de marcha lenta com variação rápida na referência.

Fonte: Autor.

Figura 96 – Controle da rotação de marcha lenta a 600 RPM.

Fonte: Autor.

3.9.4 Controle de posição da válvula borboleta eletrônica

Os gráficos a seguir mostram os resultados obtidos com o aperfeiçoamento do controle de posição da válvula borboleta eletrônica. A [Figura 97](#) exibe uma situação em que a referência de posição da borboleta varia lentamente, diferentemente da [Figura 98](#), em que a referência varia bruscamente. Nota-se que, em ambos os casos, a posição da borboleta eletrônica, dada pelo sensor TPS, acompanha fielmente a sua referência, fato que comprova a eficácia do seu controle. Além disto, não se nota a presença de oscilações na posição da borboleta, o que poderia comprometer o controle do motor.

Figura 97 – Controle de posição da válvula borboleta eletrônica com variação lenta na referência.

Fonte: Autor.

Figura 98 – Controle de posição da válvula borboleta eletrônica com variação rápida na referência.

Fonte: Autor.

Já as Figuras 99 e 100 mostram os resultados dos ensaios obtidos com a borboleta posicionada abaixo da posição de *Limp Home*, região em que o controle se mostra mais crítico em virtude do regime de marcha lenta²⁰. É importante notar que, embora aparentemente as curvas estejam mais “distantes” da referência quando comparadas às curvas anteriores (Figuras 97 e 98), o erro de posição²¹ ainda é mantido em um valor suficientemente pequeno, não superando o valor de 1% ao longo dos ensaios, fato que pode ser comprovado com base na escala do eixo vertical (posição da borboleta em porcentagem de abertura).

²⁰ Como mencionado anteriormente, durante o regime de marcha lenta a rotação do motor se torna muito sensível a pequenas variações na abertura da borboleta, dado que neste regime a abertura da borboleta já é bastante pequena, fato este que se caracteriza pela sua posição estar abaixo de *Limp Home*.

²¹ O erro de posição corresponde a diferença entre a referência e a posição real da borboleta dada pelo sensor TPS.

Figura 99 – Controle de posição da válvula borboleta eletrônica com variação lenta na referência (posição abaixo de *Limp Home*).

Fonte: Autor.

Figura 100 – Controle de posição da válvula borboleta eletrônica com variação rápida na referência (posição abaixo de *Limp Home*).

Fonte: Autor.

3.9.5 Controle da mistura ar/combustível em malha fechada (controle lambda)

As Figuras 101 e 102 mostram os resultados obtidos com a inclusão do controle da mistura a/c em malha fechada (controle lambda), realimentado por meio de um sensor lambda de banda estreita. Os diversos chaveamentos presentes no sinal deste sensor demonstram que o motor está operando com uma mistura que está muito próxima da sua relação estequiométrica ($\lambda = 1$), fato que contribui para otimizar a eficiência do catalisador.

Figura 101 – Controle lambda durante o regime de marcha lenta.

Fonte: Autor.

Figura 102 – Controle lambda em torno de 1700 RPM.

Fonte: Autor.

3.9.6 Controle de partida

A [Figura 103](#) exibe o resultado da atuação do controle de partida quando a temperatura do motor está abaixo do seu ponto de operação (motor frio). Nela é possível observar, por exemplo, o atraso de 5 segundos para liberação do controle de marcha lenta, procedimento adotado para estabilizar o motor quando operado em baixa temperatura, conforme mencionado na [subseção 3.8.1](#). Nota-se também o emprego de um ganho de injeção para permitir a partida a frio. Para evitar confusão, é importante mencionar que o valor deste ganho de injeção para a partida está embutido no fator de correção do tempo de injeção, exibido na [Figura 103](#) como “ganho de injeção”. Este fato explica o motivo pelo qual o fator de correção assume inicialmente o valor 1,3, pois neste caso este ganho de 30% inclui tanto o ganho de partida a frio de 20% como também o ganho de 10% para estratégia de motor frio. Decorridos os 5 segundos após a partida (instante de tempo $t \cong 13s$ no gráfico da [Figura 103](#)), o fator de correção é, então, reduzido para 10%, dado que a partir deste momento o ganho de partida não é mais incluído no fator de correção da injeção.

Figura 103 – Atuação do controle de partida com motor frio.

Fonte: Autor.

Já a [Figura 104](#) mostra o resultado da atuação do controle de partida quando a temperatura do motor está em torno do seu ponto de operação (motor quente). Neste caso, notam-se algumas diferenças em relação ao resultado anterior obtido com motor frio ([Figura 103](#)):

- Redução do atraso do controle de partida para apenas 1 segundo;
- Redução da abertura inicial da borboleta;
- Remoção do ganho de injeção para partida.

Vale destacar, além disto, que o controle de rotação adota inicialmente uma referência de 1100 RPM, reduzindo-a aos poucos até 800 RPM conforme o motor vai se estabilizando.

Figura 104 – Atuação do controle de partida com motor quente.

Fonte: Autor.

3.9.7 Identificação de marcha

A [Figura 105](#) mostra o resultado do algoritmo implementado na ECU para identificação da marcha engatada. No teste realizado, o motorista conduziu o veículo desde o estado de repouso até uma velocidade de aproximadamente 120 km/h, fazendo uso das 5 marchas disponíveis. Vale lembrar que quando o valor da marcha engatada se iguala a zero significa que o veículo está desengatado, situação que ocorre quando a embreagem é acionada.

Figura 105 – Identificação das marchas do veículo durante aceleração de 0 a 120 km/h.

Fonte: Autor.

3.10 Considerações Finais

O presente capítulo descreveu as tarefas realizadas na primeira fase deste trabalho, destinada ao aprimoramento do funcionamento da ECU v2.0 projetada em 2013 (PEREIRA, 2013), visando, dentre outros, a sua evolução no que diz respeito ao estado da arte de unidades de gerenciamento eletrônico de motores. Além do aprimoramento da ECU, que se deu por meio de diversas alterações no *firmware* de cada bloco do sistema, a primeira fase do trabalho também envolveu a elaboração de uma nova ferramenta de *software* que integra diversas funções, como a função de computador de bordo alternativo e a função que automatiza o ensaio de identificação do sistema formado pelo veículo em conjunto com a ECU do trabalho, tarefa necessária para o projeto do controle de cruzeiro.

Os excelentes resultados obtidos nesta primeira fase do trabalho foram decisivos para melhorar consideravelmente o funcionamento do motor, de tal modo que agora o seu comportamento está bem próximo daquele obtido quando se utiliza a sua ECU original de fábrica. Com isto, o projeto Otto IV cumpre a sua meta inicial de disponibilizar uma ECU didática para uso em futuros projetos na área de eletrônica automotiva, de tal forma

que estes passam a ter a disposição uma ECU robusta e estável para ser utilizada como plataforma de trabalho.

Além disto, esta primeira fase, ao melhorar determinadas características do sistema como, por exemplo, a dirigibilidade do veículo, pode ser compreendida como uma etapa de preparação da ECU v2.0 para receber a implementação do controle de cruzeiro, tarefa que pertence à segunda fase do trabalho e será discutida no [Capítulo 4](#).

4 Metodologia aplicada ao Projeto do Controle de Cruzeiro

A primeira fase deste trabalho, descrita no [Capítulo 3](#), teve como objetivo aprimorar o funcionamento da ECU desenvolvida em 2013 no projeto Otto II ([PEREIRA, 2013](#)), de modo a prepará-la para receber a implementação de um controle de cruzeiro (*Cruise Control*). Sendo assim, com os resultados positivos da primeira fase, o presente trabalho dá início à sua segunda fase, responsável pelo desenvolvimento deste importante recurso de assistência ao motorista (ADAS).

Esta segunda fase do trabalho se inicia com a obtenção de um modelo empírico para o sistema constituído pelo veículo em conjunto com a ECU deste trabalho, para então prosseguir com o projeto do controlador na ferramenta MATLAB. Inicialmente, a validação do controlador projetado é realizada por meio de simulações na ferramenta Simulink, as quais têm como finalidade verificar se as especificações definidas para a aplicação de controle estão sendo devidamente atendidas. Em seguida, o controlador é implementado no *firmware* da ECU v2.0, com intuito de possibilitar testes práticos em um veículo real. Vale lembrar que o veículo utilizado nos testes não possui o recurso de controle de cruzeiro em seu estado de fábrica.

Como resultado final, espera-se que o controle projetado seja capaz de manter a velocidade do veículo a um determinado valor de referência, inclusive em situações de carga. Além disto, o controle de cruzeiro deve permitir que o motorista assuma o controle da aceleração quando assim se fizer necessário (situação de ultrapassagem, por exemplo), devendo o sistema retomar automaticamente o controle da velocidade quando o pedal de aceleração for liberado.

As seções seguintes deste capítulo abordam em detalhes as tarefas referentes ao desenvolvimento do controle de cruzeiro.

4.1 Identificação do Sistema

A primeira tarefa da segunda fase do trabalho consiste na obtenção de um modelo que represente satisfatoriamente o sistema a ser controlado que, no caso do controle de cruzeiro, corresponde ao veículo equipado com a ECU deste trabalho.

A obtenção do modelo é realizada por meio de um processo de modelagem empírico denominado identificação de sistemas, o qual envolve o uso do próprio veículo em um experimento cujo controle é de responsabilidade da aplicação de monitoramento desen-

volvida na primeira fase do trabalho. Esta seção do trabalho é destinada a abordar em detalhes o processo de modelagem do sistema em questão, iniciando pelo experimento de identificação, prosseguindo com a análise dos dados obtidos, para então estimar um modelo que, ao fim do processo, é validado com auxílio dos dados do sistema real, chamado também de planta¹.

4.1.1 Sistemas dinâmicos

Segundo [Ljung \(1998\)](#), um sistema dinâmico ([Figura 106](#)) é um objeto em que variáveis interagem e produzem sinais observáveis, os quais são normalmente chamados de saídas (y). Além disto, o sistema também é afetado por estímulos externos, de tal forma que estes recebem a denominação de entradas (u) quando são passíveis de manipulação pelo observador do sistema, enquanto as perturbações, divididas em perturbações medidas (w) e perturbações não medidas (v), são caracterizadas pelos estímulos externos que não podem ser manipulados pelo observador. Vale ressaltar que as perturbações não medidas (v) somente podem ser observadas através da influência que exercem na saída do sistema ([LJUNG, 1998](#)).

Figura 106 – Diagrama de um sistema dinâmico com saída y , entrada u , perturbação medida w e perturbação não medida v .

Fonte: Adaptado de [Ljung \(1998\)](#).

O sistema geralmente é excitado pela sua entrada ($u(t)$) e pelas variáveis de perturbação ($w(t)$ e $v(t)$), embora o usuário seja capaz de manipular apenas $u(t)$. Para o processo de modelagem, os sinais de saída ($y(t)$) são variáveis importantes, uma vez que fornecem informações úteis acerca do sistema estudado. Além disto, a distinção entre entrada ($u(t)$) e perturbação medida ($w(t)$) costuma ter pouca importância em termos de modelagem do processo ([LJUNG, 1998](#)).

O fato do sistema ser referido como dinâmico significa que o valor atual de uma saída depende não somente dos valores atuais dos estímulos externos, mas também dos seus

¹ O termo planta é utilizado na literatura para se referir a um sistema que é objeto da ação de um sistema de controle ([ANGÉLICO, 2016](#)).

respectivos valores passados, bem como dos valores anteriores da própria saída (LJUNG, 1998). Isto significa que a saída do sistema em um dado instante está associada a um histórico de valores, o que caracteriza a memória de um sistema dinâmico. (NORTON, 2009).

Para o presente trabalho, o sistema dinâmico a ser modelado corresponde a um veículo Polo 2.0L em conjunto com a ECU v2.0, cujo *firmware* foi aprimorado na primeira fase deste trabalho. A inclusão da ECU do motor na modelagem do sistema em questão é importante e necessária, pois este módulo eletrônico exerce influência direta na resposta do veículo, em termos de aceleração. Além disto, para simplificar o processo de modelagem, apenas as variáveis $u(t)$ e $y(t)$ são consideradas para o sistema em questão, desprezando, portanto, os efeitos das variáveis de perturbação $w(t)$ e $v(t)$.

4.1.2 Processo de modelagem

A modelagem do sistema composto pelo veículo equipado com a ECU deste trabalho é realizada por meio de experimentação. Através de um ensaio empírico, sinais de entrada e saída são periodicamente registrados para serem posteriormente submetidos a uma análise de dados, de modo a se obter um modelo que melhor se aproxime do sistema estudado. Este processo para obtenção de um modelo com base em experimentos recebe o nome de identificação de sistemas, sendo que o modelo obtido também é conhecido por modelo caixa preta (LJUNG, 1998).

Para a construção de modelos empíricos, alterações são provocadas nas variáveis de entrada em torno de uma certa condição nominal de operação, de tal forma que a resposta dinâmica a estas excitações possibilite a determinação de um modelo para o sistema. Por esta razão, afirma-se que o processo de identificação gera modelos experimentais que são válidos apenas em uma determinada região em torno de uma condição nominal de operação (GARCIA, 2005).

Apesar deste fato que, dependendo da aplicação, pode ser visto como uma desvantagem, o processo de identificação de sistemas com base em dados experimentais dispensa a busca por modelos matemáticos gerados a partir de princípios básicos da Física, tarefa esta que pode ser complicada para processos complexos, dado que, neste caso, o modelo pode ser formado por um número elevado de equações diferenciais com um número significativo de parâmetros desconhecidos. Desta forma, gerar modelos de processos a partir de princípios fundamentais pode resultar em modelos extremamente complexos (GARCIA, 2005).

Para o processo de identificação do sistema em questão (veículo e ECU v2.0), assume-se neste trabalho que o mesmo se comporte como um sistema linear invariante no tempo (SLIT). O sistema é dito invariante no tempo quando a sua resposta a um certo sinal de entrada independe do instante em que a mesma é aplicada ao sistema, considerando as

mesmas condições iniciais (NORTON, 2009). O termo linear é utilizado quando a saída do sistema para uma combinação linear das entradas equivale a combinação linear das saídas respectivas a cada entrada individual, resultando, portanto, em um efeito de sobreposição (LJUNG, 1998).

Supondo condições iniciais nulas, um sistema do tipo SLIT pode ser descrito, dentre outras formas, pela sua função de transferência ($G(s)$) dada pela seguinte relação (OGATA, 2003):

$$G(s) = \frac{Y(s)}{U(s)} \quad (4.1)$$

em que $U(s)$ corresponde à transformada de Laplace do sinal de entrada $u(t)$ e $Y(s)$ corresponde à transformada de Laplace do sinal de saída $y(t)$. Isto revela que, conhecidos os sinais de entrada e saída, é possível se inferir uma equação para $G(s)$.

4.1.2.1 Definição de entrada e saída do modelo

A primeira tarefa do projeto do experimento para identificação de um sistema consiste na definição dos sinais que deverão ser considerados como entradas e saídas do modelo. Embora existam diferentes formas de escolha desta definição, uma regra simples consiste em adotar como entradas as variáveis que possam ser ajustadas, ao passo que as saídas sejam as variáveis que possam ser medidas, pressupondo ainda que haja uma correlação significativa entre estas variáveis (GARCIA, 2015).

Além disto, a definição de entrada e saída pode ser diretamente atrelada à aplicação de controle que será empregada posteriormente ao processo de modelagem do sistema. Neste caso, as entradas e saídas devem ser idênticas àquelas que sejam relevantes para o controlador (GARCIA, 2015). Assim, a saída $y(t)$ do modelo é definida pela variável que se deseja controlar na aplicação de controle, correspondendo, portanto, à entrada do controlador. Já a entrada $u(t)$ do modelo é definida pela variável manipulada ou de atuação, ou seja, é a variável responsável por regular a saída $y(t)$, correspondendo, portanto, à saída do controlador.

Para o presente projeto, a aplicação de controle a ser desenvolvida posteriormente ao processo de identificação do sistema corresponde ao controle de cruzeiro. Sendo assim, a saída do sistema é dada pela velocidade instantânea do veículo, variável que se deseja controlar na aplicação de controle em questão. A entrada do sistema, por sua vez, passa a corresponder à referência de torque do motor², variável a ser utilizada como sinal de atuação (ou sinal de controle) para regular a velocidade do veículo na aplicação do controle de cruzeiro.

² Vale lembrar que a referência de torque também corresponde à entrada da ECU utilizada no trabalho.

4.1.2.2 Sinal de excitação da entrada

Em ensaios de identificação de sistemas, o sinal de entrada pode influenciar fortemente na estimativa de parâmetros do modelo. Abaixo foram destacados os sinais mais utilizados em experimentos práticos (GARCIA, 2015):

- Degrau;
- RBS (*Random Binary Signal*);
- PRBS (*Pseudo Random Binary Signal*);
- GBN (*Generalized Binary Noise*);
- ruído branco filtrado;
- soma de senoides.

Devido a sua simplicidade em termos de geração de sinal, este trabalho utiliza o degrau como forma de excitação da planta. Para geração deste sinal, o usuário necessita selecionar o instante de tempo em que deve ocorrer a variação em degrau, além do valor de sua amplitude. A saída referente à este tipo de sinal, conhecida também por resposta ao degrau, fornece algumas características importantes relativas ao processo, tais como tempo de subida, sobressinal, ganho estático e constantes de tempo dominantes (GARCIA, 2015).

Para uma escolha adequada da amplitude do degrau de entrada, alguns fatores devem ser levados em consideração. Em geral, amplitudes maiores tendem a contribuir para a obtenção de modelos mais precisos, pois elevam a relação sinal/ruído, o que reduz a influência de perturbações na saída do sistema (GARCIA, 2015). Entretanto, a amplitude de entrada deve ser limitada para que as hipóteses de linearidade do sistema não sejam violadas (ISERMANN; MÜNCHHOF, 2010). Em outras palavras, a aplicação de grandes sinais na entrada poderia levar o sistema para uma região não linear, o que prejudicaria o processo de identificação.

Além disto, é importante enfatizar que outras razões de ordem prática também podem impactar na escolha da amplitude do sinal de entrada. Dependendo da natureza do processo, variações acentuadas na saída do sistema, por exemplo, podem não serem aceitas ou toleradas, o que exige limitação da entrada a um determinado valor para inibir estas condições na saída do sistema (ISERMANN; MÜNCHHOF, 2010).

No caso deste trabalho, a amplitude do degrau a ser aplicado na referência de torque do motor foi escolhida como sendo o maior valor em que não ocorre saturação do sistema, de modo a não prejudicar a sua hipótese de linearidade. Para o sistema em questão

(veículo e ECU v2.0), esta saturação é caracterizada pelo corte da injeção de combustível³ em torno de 6000 RPM, procedimento realizado pela ECU para proteger o motor. Assim, ao limitar a rotação do motor, este corte de combustível acabaria limitando também a velocidade (saída do sistema) em uma situação em que o veículo estivesse engatado.

4.1.2.3 Período de amostragem

Neste trabalho, o sistema a ser identificado é amostrado no tempo, ou seja, amostras do sinal de entrada e de saída são periodicamente registradas em pontos discretos no tempo. O intervalo de tempo entre amostras sucessivas é denominado período de amostragem (T). A taxa de amostragem (f_s), por sua vez, corresponde ao inverso do período ($f_s = 1/T$), sendo que a frequência angular de amostragem (w_s) corresponde a $2\pi/T$ (GARCIA, 2015).

A escolha do período de amostragem determina a maior componente em frequência que é possível de se observar no sinal amostrado. O teorema de Shannon estabelece que um sinal periódico necessita ser amostrado pelo menos duas vezes a cada período, ou seja, a frequência de amostragem (w_s) necessita ser pelo menos o dobro da frequência deste sinal periódico (ISERMANN; MÜNCHHOF, 2010), sendo esta conhecida por frequência de Nyquist (w_N) (LJUNG, 1998). Na prática, no entanto, é comum amostrar o sinal com uma frequência de amostragem 5 a 10 vezes superior a frequência de Nyquist (GARCIA, 2015).

Outras regras práticas são conhecidas para a escolha do período de amostragem. Uma delas, por exemplo, consiste em adotar o intervalo de amostragem $T = \tau$, sendo τ a menor constante de tempo do sistema. Esta escolha resulta em aproximadamente 6 amostras por período de um sinal periódico, o que poderia ser insuficiente, por exemplo, para um sistema com vários polos muito próximos um do outro. Por estas razões, uma escolha bastante razoável seria adotar $T = \tau/10$. Outro critério de escolha prático é assumir o período de amostragem como sendo 10% do tempo de acomodação (t_s) de uma resposta ao degrau (GARCIA, 2015).

Em termos de experimentação, é recomendável efetuar o ensaio com uma frequência de amostragem tão elevada quanto for possível, para então desprezar alguns pontos após o ensaio, se assim for necessário. Embora a adoção de uma elevada taxa de amostragem aumente consideravelmente o número de pontos coletados, este procedimento acaba evitando que o ensaio seja refeito no caso de uma possível subamostragem dos sinais, o que poderia não ser vantajoso economicamente dependendo do processo envolvido (GARCIA, 2015).

³ Este procedimento também é conhecido por *cut off*.

Desta forma, o período de amostragem escolhido para o experimento deste trabalho consiste no valor mínimo⁴ em que a aplicação de monitoramento consegue realizar a aquisição de dados da ECU sem qualquer perda ou erro.

4.1.2.4 Duração do experimento

Embora a precisão da identificação do sistema melhore com o aumento da duração do experimento, existem restrições em relação a este procedimento (GARCIA, 2015). A título de exemplo, um longo ensaio de identificação poderia contribuir para a introdução de perturbações extras no sistema, além de contribuir para uma possível variação dos parâmetros do processo (ISERMANN; MÜNCHHOF, 2010). Além disto, um ensaio muito longo pode consumir mais energia e insumos, podendo chegar ao ponto de ser inviável economicamente, dependendo do orçamento do projeto (ISERMANN; MÜNCHHOF, 2010).

Um regra prática consiste em se adotar a duração do experimento igual a 5 a 18 vezes o tempo de acomodação (t_s) do processo, sendo que este fator multiplicativo dependerá, basicamente, da relação sinal/ruído e do número de entradas do sistema. Vale ressaltar que este fator pode ser ainda menor (1 a 2 vezes) se o processo for linear e com poucas perturbações (GARCIA, 2015).

Para o ensaio de identificação do veículo, a duração foi escolhida de tal forma que a saída do sistema, dada pela velocidade, tenha tempo suficiente para atingir o seu valor de regime estacionário.

4.1.3 Ensaio de identificação automatizado

A Figura 107 ilustra o diagrama do tipo caixa preta para representação da planta em estudo⁵. Conforme mencionado nas seções anteriores, o ensaio para identificação deste sistema é realizado mediante a aplicação de um degrau na referência de torque da ECU (entrada do sistema), analisando a resposta do sistema através da medição da velocidade instantânea do veículo.

⁴ Vale lembrar que o período de amostragem mínimo corresponde, pela definição abordada, à máxima frequência de amostragem.

⁵ Lembrando que a planta, neste trabalho, corresponde ao sistema constituído pelo veículo controlado pela ECU deste trabalho.

Figura 107 – Diagrama do tipo caixa preta para representação da planta.

Fonte: Autor.

Além disto, conforme discutido na [seção 3.7](#), o ensaio é todo controlado a partir de um computador externo, por meio do *software* desenvolvido neste projeto para monitoramento da ECU. Assim, o motorista fica a cargo apenas de colocar o veículo em movimento na marcha desejada, além de segurar a sua direção durante todo o experimento⁶. Caso se faça necessário, o condutor pode abortar o ensaio de identificação a qualquer momento, acionando a embreagem, o freio, desengatando o veículo ou ainda acionando o botão de emergência.

Através dos dados definidos pelo usuário, a aplicação controla a referência de torque da ECU do motor, determinando o instante exato em que a variação do tipo degrau deve ser aplicada a esta entrada. Ao mesmo tempo, a aplicação também realiza a aquisição de dados em tempo discreto, obedecendo rigorosamente o período de amostragem definido pelo usuário antes do procedimento.

O ensaio de identificação da planta foi realizado neste trabalho com os seguintes parâmetros:

- período de amostragem: 50 milissegundos;
- duração do ensaio: 40 segundos;
- instante do degrau: 5 segundos;
- amplitude do degrau (referência de torque): 40 %;
- modo de operação da ECU: normal;
- marcha engatada no veículo: 3^a;

⁶ Vale lembrar que o experimento foi realizado com o veículo devidamente amarrado ao dinamômetro inercial.

- temperatura do motor no início do ensaio: 88,48 °C;
- carga do dinamômetro: 10 %⁷.

A Figura 108 exibe os resultados obtidos através do ensaio de identificação. Pela curva do sinal de saída $y(t)$, é possível inferir que a planta corresponde a um sistema de primeira ordem com um pequeno atraso de transporte, dado que a resposta ao degrau se assemelha a uma exponencial decrescente no tempo. Enfatiza-se, entretanto, que a confirmação desta hipótese se dará por meio do processo de análise dos dados, a ser discutido nas seções seguintes.

Figura 108 – Resposta ao degrau da planta.

Fonte: Autor.

Uma vez obtido com sucesso os dados do ensaio de identificação, a tarefa seguinte consiste na determinação de uma função de transferência para o sistema em estudo, de modo a viabilizar o projeto do controlador de cruzeiro. Além disto, esta função de transferência é designada por $G_v(s)$ ao longo das seções seguintes deste trabalho.

⁷ A carga do dinamômetro inercial foi definida com intuito de simular uma situação em que o veículo trafega em uma estrada plana.

4.1.4 Função de transferência do modelo

Conforme discutido na [subseção 4.1.3](#), o presente trabalho presume que o sistema em estudo seja de primeira ordem com atraso de transporte. Sendo assim, a sua função de transferência é dada de acordo com a seguinte relação:

$$G_v(s) = \frac{K}{\tau \cdot s + 1} \cdot e^{-\alpha \cdot s} \quad (4.2)$$

em que K corresponde ao ganho estático do sistema, τ corresponde à sua constante de tempo e α corresponde ao atraso de transporte.

Para estimar a função de transferência $G_v(s)$, utiliza-se a ferramenta MATLAB. Com os dados importados nesta ferramenta, esta tarefa pode ser realizada mediante o uso do comando *tfest*, ficando a cargo do usuário fornecer como parâmetros de entrada desta função o número de polos e zeros do modelo a ser estimado, o valor do atraso de transporte e os dados de entrada e saída do experimento.

Vale lembrar que, para facilitar o processo de identificação, o usuário pode anular as condições iniciais dos respectivos dados de entrada e saída, caso o experimento forneça medidas iniciais não nulas. No caso do presente trabalho, este procedimento se aplica ao sinal de saída, dado que a velocidade do veículo antes da aplicação do degrau não é nula. Neste caso, para anular a condição inicial da saída é necessário descontar das amostras deste sinal a velocidade inicial do veículo, que no experimento realizado equivale, na média, a 15,2 km/h.

O *script* desenvolvido em MATLAB para determinar $G_v(s)$ foi destacada a seguir:

```
%% Período de amostragem da identificação
Ts_ident = 0.05;

%% Separa os dados de interesse para a identificação
offset_velocidade = 15.2; % Valor inicial da velocidade antes do
    degrau (condição inicial)
y = ECUData.Veloc - offset_velocidade; % Saída corresponde a
    velocidade descontada a condição inicial
u = ECUData.RefTorqueECU; % Entrada do sistema corresponde à ref.
    de torque
dados_identif = iddata(y, u, Ts_ident);

.
.
.

%% Modelo do veículo: Sistema de primeira ordem com atraso de
    transporte
```

```
% Gv(s) = K*exp(-a*s) / (Tal*s + 1)]
Gv = tfest(dados_identif, 1, 0, 0.5); % Sistema de primeira ordem
    com 1 polo, nenhum zero e atraso de 0,5 segundo.
```

A variável *dados_identif* contém os dados de entrada e saída obtidos por experimentação, enquanto os parâmetros de entrada 1 e 0 indicam, respectivamente, que o sistema possui 1 polo e nenhum zero, conforme indica a função de transferência 4.2. O parâmetro 0.5 se refere ao atraso de transporte de 0,5 segundo. A função de transferência $G_v(s)$ obtida por meio do procedimento descrito é dada por:

$$G_v(s) = \frac{0,5035}{s + 0,2009} \cdot e^{-0,5 \cdot s} = \frac{2,506}{4,978 \cdot s + 1} \cdot e^{-0,5 \cdot s} \quad (4.3)$$

Comparando-se as equações 4.2 e 4.3, conclui-se facilmente que a constante de tempo τ equivale a 4,978 segundos, enquanto o ganho estático K equivale a 2,506. Além disto, é importante notar que o único polo do sistema, cujo valor corresponde a $-0,2009$, possui parte real negativa, o que indica que o modelo resultante corresponde a um sistema estável. Este fato, por sua vez, é esperado, dado que o sistema se trata de um automóvel, cuja entrada é o pedal de aceleração e a saída é a sua velocidade instantânea.

4.1.5 Validação do modelo

O comando *tfest*, além de retornar a função de transferência estimada, também fornece um parâmetro denominado *fit*. Podendo variar de 0 a 100%, este parâmetro tem como finalidade determinar o quão bem a resposta do modelo obtido se ajusta aos dados de estimativa. Isto significa que quanto mais próximo de 100% for o índice *fit*, maior será o grau de coincidência entre modelo estimado e sistema (planta) (MATHWORKS, 2016).

O índice de ajuste *fit* é determinado pela ferramenta MATLAB por meio do cálculo da raiz quadrada do erro quadrático médio. Este índice estatístico, também conhecido na literatura por *Root Mean Square Error* (RMSE), é calculado por meio da seguinte equação (MATHWORKS, 2016):

$$RMSE = \frac{\sqrt{\sum_{t=1}^N [y(t) - \hat{y}(t)]^2}}{\sqrt{\sum_{t=1}^N [y(t) - \bar{y}]^2}} = \frac{\|y(t) - \hat{y}(t)\|}{\|y(t) - \bar{y}\|} \quad (4.4)$$

sendo que $y(t)$ é o sinal de saída medido no experimento, $\bar{y}$ é o valor médio de $y(t)$ e $\hat{y}(t)$ é o sinal de saída obtido com base no modelo estimado. Com a definição do parâmetro *fit* em porcentagem, o seu cálculo é dado por (MATHWORKS, 2016):

$$fit(\%) = 100 \cdot (1 - RMSE) = 100 \cdot \left(1 - \frac{\|y(t) - \hat{y}(t)\|}{\|y(t) - \bar{y}\|}\right) \quad (4.5)$$

No caso deste trabalho, o parâmetro *fit* resultou em 99,32%, o que mostra que o modelo estimado fornece uma resposta bem próxima da resposta da planta.

O diagrama da Figura 109 foi construído na ferramenta Simulink para possibilitar uma comparação entre a resposta do modelo obtido ($G_v(s)$) e a resposta do sistema real, conferindo, deste modo, o resultado fornecido pelo parâmetro *fit*. Para permitir esta tarefa, o modelo inserido na simulação foi excitado pela mesma entrada utilizada no experimento de identificação, no caso um degrau de amplitude 40% aplicado no instante 5 segundos. Além disto, a simulação foi realizada em tempo discreto com período de amostragem definido em 50 milissegundos e duração de 40 segundos, mantendo, portanto, as mesmas condições do experimento realizado no veículo.

Figura 109 – Diagrama de simulação do modelo descrito por $G_v(s)$.

Fonte: Autor.

A Figura 110 exibe as respostas da planta e do modelo descrito pela função de transferência $G_v(s)$. Nota-se que as curvas são bem próximas, o que valida o modelo obtido pelo processo de identificação, além de conferir qualitativamente o resultado induzido a partir do parâmetro *fit*. Ressalta-se que, para facilitar a comparação dos dados, a condição inicial do sinal de saída foi anulada, ou seja, a velocidade inicial do veículo antes da aplicação do degrau foi descontada das medições experimentais da saída, possibilitando, assim, a comparação deste sinal com a resposta obtida na simulação.

Figura 110 – Comparação entre as respostas da planta e do modelo estimado.

Fonte: Autor.

4.2 Projeto do Controlador de Cruzeiro

A [seção 4.1](#) abordou a primeira parte do desenvolvimento do controle de cruzeiro, que consistiu na obtenção de um modelo empírico para o sistema composto pelo veículo equipado com a ECU deste trabalho. Como resultado, obteve-se um modelo que representa um sistema de primeira ordem com atraso de transporte, descrito pela função de transferência $G_v(s)$ dada pela equação [4.3](#).

De posse de uma função de transferência para a planta, o trabalho dá continuidade ao desenvolvimento do controle de cruzeiro através do projeto do controlador na ferramenta MATLAB. Uma vez projetado, o controlador é validado inicialmente em ambiente de simulação com auxílio da ferramenta Simulink para, em seguida, ser implementado no *firmware* da ECU v2.0, de maneira a viabilizar a sua validação final por meio de testes utilizando o automóvel Polo 2.0L em um dinamômetro inercial.

Como resultado final, espera-se que o controle projetado seja capaz de manter a velocidade do veículo a um determinado valor de referência, inclusive em situações de

carga, além de permitir que o motorista assuma o controle da aceleração quando assim se fizer necessário (situação de ultrapassagem, por exemplo).

As seções seguintes detalham as tarefas restantes para o desenvolvimento do controle de cruzeiro, incluindo a definição das especificações da resposta do controlador com base na sua aplicação em um automóvel.

4.2.1 Definição das especificações

De acordo com Ogata (2003), as características de desempenho de um sistema de controle são frequentemente especificadas em termos da sua resposta transitória a uma entrada do tipo degrau, dado que este tipo de entrada pode ser facilmente gerada e ainda pode ser utilizada para determinar matematicamente a resposta do sistema a outros tipos de entradas.

Em geral, a resposta ao degrau de um sistema de controle prático apresenta oscilações amortecidas no regime transitório, antes de atingir o valor de regime do estado estacionário (OGATA, 2003), conforme ilustra a Figura 111. Geralmente, esta resposta é caracterizada pelos seguintes parâmetros (OGATA, 2003):

- Tempo de atraso t_d : Tempo necessário para que a saída do sistema alcance metade do seu valor final (conhecido também por valor de regime estacionário);
- Tempo de subida t_r : Tempo necessário para que a saída do sistema varie de 10% a 90%, de 5% a 95%, ou ainda de 0% a 100% do seu valor final (tipicamente se adota 10% a 90%);
- Tempo (ou instante) de pico t_p : Instante em que a saída do sistema atinge o seu valor máximo, conhecido também como pico de sobressinal;
- Máximo sobressinal (*overshoot*) M_p : Corresponde ao valor de pico da saída do sistema ($c(t_p)$) medido a partir do valor de regime estacionário ($c(\infty)$), de acordo com a seguinte equação:

$$M_p(\%) = \frac{c(t_p) - c(\infty)}{c(\infty)} \times 100\% \quad (4.6)$$

- Tempo de estabilização ou acomodação t_s : Tempo necessário para que a saída do sistema alcance e permaneça dentro de uma faixa de tolerância (geralmente 2% ou 5%) em relação ao seu valor final.

Figura 111 – Resposta transitória ao degrau de um sistema de controle.

Fonte: Ogata (2003).

Para o controle de cruzeiro proposto neste trabalho, as especificações da resposta do sistema são dadas em termos do máximo sobressinal (M_p) e do tempo de estabilização (t_s) permitidos para a saída, sendo que a escolha dos valores destes parâmetros está diretamente atrelada à aplicação do controle de cruzeiro.

O máximo sobressinal permitido para a saída do sistema deve ser escolhido como sendo o menor possível, de maneira que o condutor do veículo não seja penalizado com uma possível infração de trânsito por excesso de velocidade ao acionar o controle de cruzeiro (regime transitório). Segundo a [Companhia de Engenharia de Tráfego \(2017\)](#), a tolerância dos radares está associada a uma margem de erro admitida para os velocímetros do veículo, sendo que o valor máximo permitido, conforme a Portaria nº115 do INMETRO, corresponde 7 km/h para velocidades até 100 km/h e 7% para velocidades acima de 100 km/h. Levando em conta estas restrições, adota-se 7% para o sobressinal máximo da resposta do sistema ($M_p = 7\%$).

O tempo de estabilização, por sua vez, não deve ser um valor muito pequeno, de tal modo que o controlador seja capaz de corrigir suavemente a velocidade até o valor desejado, chamado também de referência ou *setpoint*. Em outras palavras, forçar um tempo de estabilização muito reduzido iria resultar em um sinal de controle muito elevado

por parte do controlador, o que resultaria em um grande desconforto aos passageiros em virtude da aceleração exacerbada desenvolvida pelo veículo.

Ao mesmo tempo, um tempo de estabilização muito elevado resultaria em um grande retardado na resposta do controlador e, neste caso, a velocidade levaria muito tempo para alcançar o valor de referência, o que também acaba sendo indesejável para a aplicação de controle em questão. Portanto, considerando estas restrições e demais aspectos de ordem prática, adota-se inicialmente um tempo de estabilização de 4 segundos ($t_s = 4s$).

Por fim, é importante mencionar que as especificações definidas para o controlador de cruzeiro são validadas, por simplicidade, com o veículo trafegando em pista plana com terceira marcha engatada. Isto significa que em uma descida acentuada, por exemplo, fica a cargo do condutor controlar a velocidade do veículo por meio de atuação no freio. Ao mesmo tempo, é possível que em uma subida o tempo de estabilização seja maior do que o valor especificado, o que não representa um problema para a aplicação em questão.

4.2.2 Projeto do controlador

Visando atender as especificações definidas, o trabalho adotou o controlador *Dahlin* para o projeto do controle de cruzeiro. Este controlador, considerado como uma modificação do controlador *Deadbeat*, assume que o sistema em malha fechada se comporta como um sistema de primeira ordem com atraso de transporte, de tal forma que a função de transferência em malha fechada, designada por $G_{MF}(s)$, possa ser representada pela seguinte relação (ANGÉLICO, 2016):

$$G_{MF}(s) = \frac{e^{-a \cdot s}}{q \cdot s + 1} \quad (4.7)$$

No plano Z, assume-se que o atraso de transporte é um múltiplo do período de amostragem, ou seja, $a = k \cdot T_s$. Assim, levando-se em conta ainda o efeito de um segurador de ordem zero (*Zero-Order Hold* - ZOH) na amostragem, a forma discreta da equação 4.7 passa a ser dada por (ANGÉLICO, 2016):

$$G_{MF}(z) = \frac{z^{-k-1} \cdot (1 - e^{-T_s/q})}{1 - e^{-T_s/q} \cdot z^{-1}} \quad (4.8)$$

A Figura 112 mostra o diagrama de um sistema operando em malha fechada com compensação de um controlador $C(z)$. Neste caso, a função de transferência em malha fechada pode ser escrita como:

$$G_{MF}(z) = \frac{Y(z)}{R(z)} = \frac{G(z) \cdot C(z)}{1 + G(z) \cdot C(z) \cdot H(z)} \quad (4.9)$$

Isolando o termo $C(z)$ e assumindo realimentação unitária ($H(z) = 1$), obtém-se a seguinte expressão para a função de transferência do controlador:

$$C(z) = \frac{1}{G(z)} \cdot \frac{G_{MF}(z)}{1 - G_{MF}(z)} \quad (4.10)$$

Figura 112 – Diagrama de um sistema em malha fechada com compensação.

Fonte: Autor.

Conclui-se, com base na equação 4.10, que a função de transferência do controlador $C(z)$ pode ser determinada com auxílio da função de transferência da planta $G(z)$ e também com a função de transferência do sistema em malha fechada $G_{MF}(z)$ que, no caso do controlador *Dahlin*, pode ser calculada por meio da equação 4.8. É importante notar, no entanto, que o cálculo de $G_{MF}(z)$ depende basicamente de três parâmetros: período de amostragem T_s ⁸, constante de tempo da resposta do sistema em malha fechada q e da constante k .

Conforme mencionado, a estrutura do controlador *Dahlin* força que o sistema em malha fechada se comporte como um sistema de primeira ordem com atraso. Neste caso, a constante de tempo q pode ser relacionada com o tempo de estabilização t_s definido nas especificações do projeto do controlador. O gráfico da Figura 113, por exemplo, sugere que após decorrido um intervalo de tempo equivalente à 5 constantes de tempo a saída do sistema terá alcançado 99,3% do seu valor de regime, o que já é aceitável para a faixa de estabilidade da resposta. Portanto, a constante de tempo q pode ser aproximada por:

$$q \cong \frac{t_s}{5} = 0,8s \quad (4.11)$$

⁸ Não se deve confundir o período de amostragem T_s aqui mencionado com o período de amostragem utilizado no ensaio de identificação da planta, abordado na seção 4.1.

Figura 113 – Relação entre constante de tempo e valor de regime para um sistema de primeira ordem.

Fonte: Ogata (2003).

A escolha do período de amostragem pode ser baseada em regras práticas, conforme abordado na [subseção 4.1.2.3](#). Uma delas consiste, por exemplo, em adotar o valor do período de amostragem como sendo 10% do valor da menor constante de tempo do sistema. Neste caso, em que o sistema é de primeira ordem, tem-se:

$$T_s \cong 0,1 \cdot q = 0,08s = 80ms \quad (4.12)$$

Entretanto, para facilitar a posterior implementação do controlador, adota-se, em vez de 80 milissegundos, o valor de 100 milissegundos para o período de amostragem do controlador:

$$T_s = 0,1s = 100ms \quad (4.13)$$

O parâmetro k deve ser escolhido para manter a causalidade do controlador $C(z)$ e depende, portanto, da função de transferência da planta $G(z)$. Para o presente projeto, a função de transferência discreta da planta pode ser obtida por meio da discretização da equação 4.3, tarefa que pode ser realizada com auxílio do comando `c2d` da ferramenta MATLAB:

```

%% Especificacoes da resposta do controlador
Ts = 0.1; % Período de amostragem do controle
ts = 4.0; % Tempo de estabilização (em s) da resposta em malha
        fechada
q = ts / 5; % Constante de tempo da resposta (primeira ordem)

%% Discretização da planta: Gv(s) -> G(z)
z = tf('z', Ts);
G = c2d(Gv, Ts, 'zoh');

```

O resultado obtido para $G(z)$ é dado pela seguinte função de transferência discreta:

$$G(z) = z^{-5} \cdot \frac{0,04985 \cdot z^{-1}}{1 - 0,9801 \cdot z^{-1}} = \frac{0,04985}{z^6 - 0,9801 \cdot z^5} \quad (4.14)$$

É importante notar que a discretização da planta não prejudicou a estabilidade do sistema, dado que todos os polos de $G(z)$ possuem módulo menor do que 1 e, portanto, estão localizados estritamente dentro do círculo de raio unitário no plano Z, condição necessária para a estabilidade de um sistema discreto (ANGÉLICO, 2016). Além disto, verifica-se, por meio de uma rápida análise da equação 4.14, que a planta possui 6 zeros no infinito, o que significa que qualquer $k \geq 5$ é condição suficiente para garantir a causalidade do controlador $C(z)$. Adota-se, portanto, $k = 5$.

A seguir, foram destacados os parâmetros definidos para a função de transferência de malha fechada $G_{MF}(z)$:

- constante de tempo q : 0,8 segundo;
- período de amostragem T_s : 0,1 segundo;
- parâmetro k : 5.

Substituindo estes parâmetros na equação 4.8, resulta:

$$G_{MF}(z) = \frac{0,1175}{z^6 - 0,8825 \cdot z^5} \quad (4.15)$$

Nota-se que, similarmente à $G(z)$, a função de transferência $G_{MF}(z)$ possui todos os seus polos localizados estritamente dentro do círculo de raio unitário no plano Z, o que significa que o sistema em malha fechada também é estável. Substituindo os resultados das equações 4.15 e 4.14 na equação 4.10, a função de transferência do controlador $C(z)$ passa a ser dada por:

$$C(z) = \frac{2,357 - 2,31 \cdot z^{-1}}{1 - 0,8825 \cdot z^{-1} - 0,1175 \cdot z^{-6}} = \frac{2,357 \cdot z^6 - 2,31 \cdot z^5}{z^6 - 0,8825 \cdot z^5 - 0,1175} \quad (4.16)$$

Figura 114 – Lugar das raízes do sistema compensado com $C(z)$.

Fonte: Autor.

A Figura 114 destaca o lugar das raízes da planta $G(z)$ compensada com o controlador $C(z)$.

O seguinte script em MATLAB foi utilizado para realizar o projeto de $C(z)$:

```
%% Projeto controlador digital pelo método direto (Controle
    Dahlin):
% C(z) = 1/G(z)*G_MF(z)/(1-G_MF(z))
% Com G_MF(z) = (1 - exp(-Ts/q))*z^(-k-1)/(1 - exp(-Ts/q)*z^-1)
% Dado que G(z) tem 6 zeros no infinito, necessita-se k >= 5 para
    garantir
% a causalidade. Adota-se, portanto, k = 5.
k = G.iodelay; % k = 5
G_MF = (1 - exp(-Ts/q))*z^(-k-1)/(1 - exp(-Ts/q)*z^-1); % FT de
    malha fechada
G_MF = minreal(G_MF); % Simplifica G_MF(z)
%inv_G = 1/G; % Inversa de G(z) - Matlab aceita somente se G(z) n
    ão tiver atraso!
inv_G = z^G.iodelay * tf(G.den, G.num, Ts); % Inversa de G(z)
    para o caso com atraso!
C =(inv_G*G_MF)/(1-G_MF); % Controlador digital projetado
C = minreal(C); % Simplifica C(z)
```

4.2.3 Validação do controlador por meio de simulação

Com auxílio da ferramenta Simulink, a validação inicial do controle de cruzeiro é realizada por meio de simulações, visando verificar se as especificações definidas para o controlador estão sendo devidamente respeitadas.

A [Figura 115](#) exibe o diagrama de simulação construído na ferramenta Simulink para validar o controlador $C(z)$ projetado. A seguir foram listados os principais sinais presentes no diagrama:

- $r(t)$: velocidade de referência do controle de cruzeiro;
- $r(n)$: velocidade de referência em tempo discreto (amostragem de $r(t)$);
- $y(t)$: saída do sistema, correspondente à velocidade do veículo;
- $y(n)$: saída do sistema em tempo discreto (amostragem de $y(t)$);
- $u(n)$: saída do controlador em tempo discreto, chamada também de sinal de controle;
- $e(n)$: erro de velocidade em tempo discreto, correspondente à diferença entre referência e velocidade ($e(n) = r(n) - y(n)$);

Figura 115 – Diagrama de simulação do controle de cruzeiro.

Fonte: Autor.

A fim de aproximar o ambiente de simulação do cenário real, os itens seguintes foram levados em conta para construir o diagrama da [Figura 115](#):

1. Embora o controlador tenha sido projetado em tempo discreto, a planta utilizada na simulação corresponde a de tempo contínuo ($G_v(s)$), para que seja possível verificar se a saída contínua $y(t)$ apresenta oscilações entre os instantes de amostragem.
2. Seguradores de ordem zero (ZOH) são empregados na simulação para converter sinais de tempo contínuo em sinais de tempo discreto, e vice-versa, representando assim o papel de um conversor A/D (ou D/A) na aplicação real.
3. Ruído branco é adicionado à saída do sistema para contabilizar o ruído inerente à medição da velocidade do veículo.
4. Um distúrbio em degrau é aplicado à saída do sistema em regime estacionário para avaliar a robustez do controlador nesta situação.
5. A saída do controlador é limitada por um bloco de saturação, com intuito de respeitar as limitações da entrada da planta (referência de torque da ECU).

Os parâmetros da simulação foram definidos como segue:

- Período de amostragem dos blocos ZOH: 100 milissegundos, igual ao período de amostragem T_s do controlador;
- Duração da simulação: 100 segundos;
- Instante da variação em degrau na referência: 30 segundos;
- Instante da aplicação do distúrbio na saída do sistema: 70 segundos;

- Valores limites do bloco de saturação da saída do controlador: 0 a 100%;
- Variância do ruído aplicado à saída do sistema: $0,01 (km/h)^2$;
- Valor inicial e final do degrau aplicado à referência: 40 a 50 km/h, 50 a 80 km/h e 50 a 120 km/h, totalizando, portanto, três simulações diferentes;
- Amplitude do distúrbio em degrau aplicado à saída: definido igual à amplitude do degrau aplicado à referência e, portanto, assume os valores 10, 30, e 70 km/h.

Vale ressaltar que os respectivos valores utilizados no degrau da referência são apenas para fins de simulação, pois, na prática, a ECU atribui à referência o valor da velocidade do veículo medida no instante em que o condutor aciona o controle de cruzeiro. Isto significa que, diferentemente das simulações, o controlador, neste caso prático, recebe como entrada um erro inicial nulo ou mínimo quando o controle de cruzeiro é acionado (regime transitório).

Além disto, é importante observar no diagrama da [Figura 115](#) que o distúrbio é aplicado à saída por meio de uma operação de subtração, para simular uma situação em que o mesmo atua no sentido de reduzir a velocidade do veículo.

A [Figura 116](#) traz o resultado da primeira simulação, caracterizada pela aplicação de um degrau de 40 a 50 km/h na referência do sistema em malha fechada, além de um distúrbio de amplitude 10 km/h aplicado à saída do sistema em regime estacionário. Nota-se que o controlador reagiu rapidamente tanto à variação em degrau de $r(t)$ no instante 30 segundos como também ao distúrbio aplicado à saída $y(t)$ no instante 70 segundos, corrigindo-a em menos de 4 segundos, o que está de acordo com a especificação do projeto que exige tempo de estabilização máximo de 4 segundos. Observa-se, além disto, que a saída não apresentou sobressinal, o que atende a outra especificação do projeto que exige sobressinal máximo de 7%.

Figura 116 – Simulação do controle de cruzeiro com degrau na referência de 40 a 50 km/h.

Fonte: Autor.

A Figura 117 exibe as curvas obtidas na primeira simulação (Figura 116), porém com ampliação em torno do instante 30 segundos. Este procedimento é realizado para evidenciar que a saída contínua $y(t)$ não apresentou oscilações entre os instantes de amostragem, característica desejável para uma aplicação de controle. Uma vez constatado este fato, as simulações seguintes omitem a curva da saída discreta $y(n)$, com intuito de se obter uma melhor visualização da saída em tempo contínuo $y(t)$.

Figura 117 – Ausência de oscilações entre os instantes de amostragem da saída (simulação 40-50 km/h).

Fonte: Autor.

A simulação seguinte, caracterizada pela aplicação de degrau de amplitude 50-80 km/h na referência, também apresentou resposta em conformidade com as especificações do projeto do controlador, fato que pode ser constatado com base nas curvas da [Figura 118](#).

Figura 118 – Simulação do controle de cruzeiro com degrau na referência de 50 a 80 km/h.

Fonte: Autor.

A Figura 119 ilustra o resultado da última simulação, caracterizada pela aplicação de um degrau 50-120 km/h na referência, além da aplicação de um distúrbio de 70 km/h na saída do sistema em regime estacionário. Esta simulação ilustra uma situação atípica em que o sinal de controle $u(n)$ chega a saturar em seu valor máximo de 100% devido à elevada amplitude do degrau aplicado em $r(t)$, como mostram as curvas da Figura 119. A presença de sobressinal e o elevado tempo de acomodação evidenciam que a resposta, neste caso especial, não seguiu as especificações de projeto, fato que é esperado uma vez que a saturação do sinal de controle compromete a hipótese de linearidade do sistema nesta condição de operação.

Entretanto, esta situação não deve ocorrer na prática com o controlador de cruzeiro implementado em *firmware*, pois a referência, neste caso, é definida com o valor da velocidade no instante do acionamento, o que significa que o controlador é acionado com erro inicial nulo ou mínimo, o que evita a saturação do sinal de controle. Além disto, adianta-se que o trabalho, no intuito de reforçar a segurança do sistema, busca corrigir o sobressinal presente na resposta do sistema nesta condição especial, através

da implementação do controlador por meio de sua equação de diferenças, tarefa a ser discutida na [subseção 4.2.4](#).

Figura 119 – Simulação do controle de cruzeiro com degrau na referência de 50 a 120 km/h.

Fonte: Autor.

4.2.4 Simulação com o controlador implementado em *Matlab Function*

A [subseção 4.2.3](#) abordou a validação inicial do controlador $C(z)$ através de simulações. O diagrama de simulação utilizado neste caso ([Figura 115](#)) foi construído em Simulink fazendo uso do bloco *LTI System* para representar o controlador $C(z)$ e a planta $G_v(s)$ por meio de suas respectivas funções de transferência.

Nesta subseção, o trabalho destaca uma segunda validação em que se emprega, em vez do bloco *LTI System*, o bloco *Matlab Function* para representar o controlador $C(z)$ no diagrama de simulação, tarefa essencial para facilitar a sua posterior implementação em *firmware*.

O bloco *Matlab Function* permite o uso de funções do MATLAB em modelos de simulação criados em Simulink, viabilizando, deste modo, a implementação destas funções

em processadores e microcontroladores. Como resultado, este recurso se revela bastante útil para os casos em que a simulação envolve algoritmos de codificação que são melhores descritos na linguagem textual do MATLAB do que na linguagem gráfica do Simulink (MATHWORKS, 2017). Por esta razão, a simulação com *Matlab Function* tende a fornecer resultados mais próximos do cenário real, em que o controlador é implementado com auxílio da eletrônica embarcada, fato que revela a importância deste bloco para validação do controle projetado.

No caso do projeto do controle de cruzeiro, este bloco é responsável por implementar em código MATLAB a equação de diferenças do controlador, esta que, por sua vez, pode ser obtida através da função de transferência $C(z)$, dada pela equação 4.16. Se $e(n)$ e $u(n)$ são, respectivamente, os sinais de entrada e saída do controlador, pode-se denotar por $E(z)$ e $U(z)$ as respectivas transformadas Z destes sinais. Com isto, a função de transferência $C(z)$ pode ser reescrita como:

$$C(z) = \frac{U(z)}{E(z)} = \frac{2,357 - 2,31 \cdot z^{-1}}{1 - 0,8825 \cdot z^{-1} - 0,1175 \cdot z^{-6}} \quad (4.17)$$

$$\iff U(z) = 0,8825 \cdot z^{-1} \cdot U(z) + 0,1175 \cdot z^{-6} \cdot U(z) + 2,357 \cdot E(z) - 2,31 \cdot z^{-1} \cdot E(z) \quad (4.18)$$

A propriedade do atraso da transformada Z afirma que atrasar uma sequência no tempo por n_0 amostras é equivalente a multiplicar a sua transformada por z^{-n_0} , ou seja (ANGÉLICO, 2016):

$$x(n - n_0) \xrightarrow{Z} z^{-n_0} \cdot X(z) \quad (4.19)$$

Aplicando esta propriedade para calcular a anti-transformada Z de 4.18, a equação de diferenças do controlador resulta em:

$$u(n) = 0,8825 \cdot u(n - 1) + 0,1175 \cdot u(n - 6) + 2,357 \cdot e(n) - 2,31 \cdot e(n - 1) \quad (4.20)$$

A Figura 120 exibe o diagrama de simulação construído na ferramenta Simulink com uso do bloco *Matlab Function* para implementar o controlador de cruzeiro. É importante mencionar que este diagrama foi construído com base no diagrama da Figura 115, procedimento este que acaba preservando os parâmetros do ambiente de simulação.

Figura 120 – Diagrama de simulação do controle de cruzeiro com uso do bloco *Matlab Function* para implementar o controlador $C(z)$.

Fonte: Autor.

Abaixo foi destacado o código do bloco *Matlab Function* que implementa a equação de diferenças 4.20 do controlador. É importante notar que a saturação do sinal de controle foi levada em conta na implementação deste código, fato que justifica a ausência do bloco de saturação no diagrama da Figura 120.

```
function u = fcn(e)
%#codegen
global e_ant;
global u_ant;
global u_ant2;
global u_ant3;
global u_ant4;
global u_ant5;
global u_ant6;

% Definição das limitações do sinal de controle
u_max = 100;
u_min = 0;

% Cálculo do sinal de controle (controlador Dahlin)
u_temp = 0.8825*u_ant + 0.1175*u_ant6 + 2.357*e - 2.31*e_ant; % C
(z)
```

```
% Limitação do sinal de controle
if (u_temp > u_max)
    u_temp = u_max;
elseif (u_temp < u_min)
    u_temp = u_min;
end

% Deslocamento das amostras para o próximo laço de controle
e_ant = e;
u_ant6 = u_ant5;
u_ant5 = u_ant4;
u_ant4 = u_ant3;
u_ant3 = u_ant2;
u_ant2 = u_ant;
u_ant = u_temp;

% Passa o sinal calculado para a saída do controlador
u = u_temp;
```

Para efeito de comparação, as mesmas três simulações da [subseção 4.2.3](#) são novamente realizadas com o controlador implementado por meio da sua equação de diferenças. As Figuras [121](#) e [122](#) exibem, respectivamente, os resultados obtidos nas simulações com degrau na referência de 40-50 km/h e 50-80 km/h. Conclui-se, por meio de uma rápida comparação, que estes resultados são idênticos aos respectivos resultados obtidos nas simulações da [subseção 4.2.3](#) (Figuras [116](#) e [118](#)), o que valida a implementação do controlador por meio da sua equação de diferenças.

Figura 121 – Simulação do controle de cruzeiro com degrau na referência de 40 a 50 km/h (*Matlab Function*).

Fonte: Autor.

Figura 122 – Simulação do controle de cruzeiro com degrau na referência de 50 a 80 km/h (*Matlab Function*).

Fonte: Autor.

Além disto, da mesma forma que foi feito na subseção 4.2.3, a Figura 123 traz uma ampliação das curvas da primeira simulação (Figura 121) em torno do instante 30 segundos, com intuito de mostrar que a saída contínua $y(t)$ não apresentou oscilações entre os instantes de amostragem.

Figura 123 – Ausência de oscilações entre os instantes de amostragem da saída (simulação 40-50 km/h com *Matlab Function*).

Fonte: Autor.

Por fim, a Figura 124 traz o resultado da última simulação realizada com degrau 50-120 km/h na referência do sistema. Comparando este resultado com a respectiva simulação da Figura 119, é possível observar que agora a resposta não apresentou sobressinal, fato que favorece a segurança do sistema. Além disto, nota-se que o tempo de acomodação da resposta aumentou consideravelmente se comparado à simulação anterior, fato que, no entanto, é aceitável em virtude do sistema estar operando em uma região não linear limitada pelo torque máximo do motor.

É importante ressaltar que a supressão do sobressinal nesta última simulação (Figura 124) se deve basicamente ao emprego de limitação para o sinal de controle calculado no código MATLAB, sendo esta limitação imposta antes do sinal de controle ser memorizado pelo algoritmo, ou seja, antes das amostras do sinal de controle serem deslocadas no tempo ($u(n-6) = u(n-5)$; $u(n-5) = u(n-4)$; ... $u(n-1) = u(n)$). Como resultado, evita-se que o sinal de controle seja memorizado com valores elevados e acima da saturação (*anti-windup*).

Figura 124 – Simulação do controle de cruzeiro com degrau na referência de 50 a 120 km/h (*Matlab Function*).

Fonte: Autor.

4.2.5 Implementação do controlador em *firmware*

A subseção 4.2.4 abordou uma segunda validação do controle de cruzeiro em que se emprega o bloco *Matlab Function* para implementar o controlador no diagrama de simulação. Escrito em linguagem MATLAB, o código deste bloco foi desenvolvido tomando por base a equação de diferenças do controlador projetado.

Para possibilitar a implantação do controlador no *firmware* do bloco de Gerenciamento da ECU v2.0, é necessário reescrever em linguagem C o código MATLAB do bloco do controlador, o que é uma tarefa relativamente simples, em virtude da semelhança de sintaxe destas linguagens. Entretanto, a implementação final do controle de cruzeiro na ECU deve levar em conta alguns aspectos de ordem prática, como a definição da velocidade de referência, as condições de segurança para desativar o controle e a retomada da aceleração pelo condutor. A Figura 125 ilustra o digrama de blocos do controle de cruzeiro introduzido na ECU deste trabalho.

Figura 125 – Diagrama de blocos do controle de cruzeiro.

Fonte: Autor.

As seções seguintes abordam os principais recursos embutidos na implementação final do controle de cruzeiro, sendo que a [subseção 4.2.5.3](#) disponibiliza o fluxograma da implementação em *firmware*. Vale lembrar ainda que o [Apêndice A](#) disponibiliza a implementação em *firmware* do controle de cruzeiro desenvolvido neste trabalho.

4.2.5.1 Características do acionamento do controle de cruzeiro

O acionamento do controle de cruzeiro é realizado através de um botão “liga desliga” ([Figura 126](#)), instalado no painel do veículo exclusivamente para esta finalidade. As seguintes ações são tomadas pelo *firmware* no instante em que o controle é acionado:

1. Definição da velocidade de referência com o valor da velocidade do veículo medida no instante em que o controle é acionado;
2. Determinação de ajuste empírico a ser aplicado como valor mínimo do sinal de controle inicial, necessário para que a velocidade do veículo não apresente queda logo após o controle ser acionado (correção do regime transitório);
3. Ativação de um *timer* para medição de 5 segundos, janela de tempo em que o condutor está impossibilitado de retomar o controle da aceleração do veículo através do pedal de aceleração.

Uma vez em funcionamento, o controle de cruzeiro é desativado pela ECU quando ocorrer pelo menos um dos seguintes eventos:

- Retorno do botão para a posição desligado;
- Acionamento do pedal da embreagem;
- Acionamento do pedal do freio;
- Desengate do veículo;
- Erro de *hardware* (bloco de Gerenciamento).

Figura 126 – Botão de acionamento do controle de cruzeiro da ECU v2.0.

Fonte: Autor.

Ressalta-se que na situação em que o controle de cruzeiro é desativado sem fazer uso do botão do painel, o condutor necessita, por medida de segurança, retornar o respectivo botão para a posição desligado para conseguir reativar o controle de cruzeiro. Além disto, é importante mencionar que, também como medida de segurança, o controle de cruzeiro não opera em primeira marcha, tampouco quando a velocidade do veículo está abaixo de 10 km/h.

4.2.5.2 Retomada da aceleração do veículo para ultrapassagem

Conforme mencionado no início deste capítulo, o controle de cruzeiro proposto neste trabalho deve estar preparado para permitir que o motorista assuma o controle da aceleração quando for necessário, devendo o sistema retomar automaticamente o controle da velocidade quando o pedal de aceleração for liberado, evitando, portanto, que o motorista tenha que reativar manualmente o controle. Este recurso se mostra bastante útil em situações em que se necessita de uma aceleração rápida como, por exemplo, em uma ultrapassagem.

Em termos de implementação, a malha do controle de cruzeiro verifica, a cada laço de execução (100 milissegundos), se a referência de torque equivalente à posição do pedal de aceleração é maior do que o sinal de controle calculado pelo controlador Dahlin. Em caso afirmativo, a referência de torque da ECU é definida com o valor dado pelo pedal, em vez do sinal de controle calculado pelo controlador, sendo este último memorizado pelo *firmware* para ser utilizado posteriormente quando o pedal de aceleração for liberado pelo condutor.

Vale lembrar que o controle de cruzeiro deste trabalho permite que o condutor faça uso do pedal de aceleração somente 5 segundos após o controle ser acionado, janela de tempo necessária para que o sistema alcance o regime estacionário.

4.2.5.3 Fluxograma da implementação em *firmware* do controle de cruzeiro

A [Figura 127](#) apresenta o fluxograma da implementação em *firmware* do controle de cruzeiro desenvolvido, o qual inclui, além do próprio controlador Dahlin, os recursos abordados nas seções anteriores.

Figura 127 – Fluxograma do controle de cruzeiro.

4.3 Resultados

Uma vez que o controlador de cruzeiro é implementado no *firmware* da ECU v2.0, o trabalho prossegue com a sua validação por meio de testes aplicando o veículo Polo 2.0L em um dinamômetro inercial, de tal forma que seja possível estabelecer uma comparação com os resultados obtidos nas simulações, além de permitir avaliar na prática a robustez do controlador. Esta seção, portanto, destaca os resultados obtidos nesta validação prática do controle de cruzeiro.

É importante ressaltar que o valor da carga do dinamômetro, embora não esteja presente nos gráficos devido à razões técnicas, está diretamente relacionado ao valor da referência de torque e ao valor do consumo de combustível, ambos presentes nos gráficos seguintes. Desta forma, quando o controle de cruzeiro estiver acionado, um aumento da carga do dinamômetro refletirá em um aumento da referência de torque e, ao mesmo tempo, em um aumento do consumo de combustível⁹.

4.3.1 Controle de cruzeiro acionado a 50 km/h

A [Figura 128](#) destaca o resultado do primeiro teste prático, em que o controle de cruzeiro foi acionado a 50 km/h com o veículo engatado em terceira marcha. Neste teste, a carga do dinamômetro foi lentamente aumentada de 0 a 30% para, em seguida, ser reduzida novamente até 0, possibilitando, deste modo, avaliar o comportamento do controlador na presença de perturbação, representada neste caso pela carga do dinamômetro. Pelas curvas de referência de torque e de consumo, é possível observar que o início da variação da carga do dinamômetro ocorreu em torno do instante 10 segundos, sendo que o valor máximo de 30% foi alcançado por volta de 37 segundos. Em seguida, por volta do instante 47 segundos, a carga começou a ser reduzida lentamente até seu valor mínimo, atingido-o no instante 75 segundos.

De modo geral, observa-se que, ao longo do teste, a velocidade do veículo permaneceu estável e em torno do valor desejado (50 km/h), apresentando apenas um pequeno erro transitório durante a aplicação da carga, o que valida a operação do controle de cruzeiro na presença de perturbação. Nota-se também que este erro transitório não ultrapassou 3 km/h, o que é totalmente aceitável para a aplicação em questão.

⁹ É importante notar que um aumento do consumo de combustível se traduz em uma redução da sua medida em km/l. Isto significa que um veículo indicando um consumo de 10 km/l, por exemplo, está consumindo menos combustível do que outro indicando 7 km/l.

Figura 128 – Controle de cruzeiro acionado a 50 km/h com variação lenta na carga do dinamômetro.

Fonte: Autor.

A [Figura 129](#) destaca o resultado do segundo teste, realizado também com o controle de cruzeiro acionado a 50 km/h, mas com variação em degrau de 0 a 20% na carga do dinamômetro. Pelas curvas de referência de torque e de consumo, é possível observar que o degrau na carga foi aplicado em torno de 12 segundos, enquanto o degrau oposto (20 a 0%) ocorreu por volta de 42 segundos. Além disto, observa-se que, mesmo com uma variação extremamente rápida da carga, a velocidade se manteve estável e apresentou um erro transitório que não ultrapassou 5 km/h, o que confirma a robustez do controle de cruzeiro na operação com carga. Vale lembrar que esta situação de variação em degrau na carga é extremamente rara de ocorrer no caso em que o veículo trafega em pista, pois neste caso a carga aplicada ao sistema está diretamente atrelada à inclinação da pista, cujo valor não costuma apresentar variação brusca.

Figura 129 – Controle de cruzeiro acionado a 50 km/h com variação rápida na carga do dinamômetro.

Fonte: Autor.

4.3.2 Controle de cruzeiro acionado a 100 km/h

Os mesmos testes realizados a 50 km/h foram novamente executados com o controle de cruzeiro acionado a 100 km/h, de maneira a se aproximar de uma situação em que o veículo trafega em uma pista de maior velocidade, a exemplo de uma rodovia. As Figuras 130 e 131 ilustram, respectivamente, os resultados obtidos com variação lenta e rápida (degrau) na carga do dinamômetro. É importante mencionar que no primeiro teste a carga foi lentamente variada entre 0 e 25%, enquanto no segundo teste foram mantidos os valores limites de 0 e 20% para o degrau da carga.

Pelas curvas obtidas, observa-se que o controle de cruzeiro apresentou um erro transitório de valor máximo inferior a 4 km/h no caso de variação lenta na carga (Figura 130), e inferior a 7 km/h no pior caso em que a carga foi aplicada em degrau (Figura 131), fatos estes que representam ótimos resultados para a aplicação em questão.

Figura 130 – Controle de cruzeiro acionado a 100 km/h com variação lenta na carga do dinamômetro.

Fonte: Autor.

Figura 131 – Controle de cruzeiro acionado a 100 km/h com variação rápida na carga do dinamômetro.

Fonte: Autor.

4.3.3 Uso do pedal de aceleração com o controle de cruzeiro acionado

Conforme mencionado, o controle de cruzeiro deve permitir que o motorista faça uso do pedal de aceleração quando for necessário e, neste caso, o sistema deve retomar automaticamente o controle da velocidade quando o pedal de aceleração for liberado. Para demonstrar esta funcionalidade, foi realizado o teste destacado na [Figura 132](#).

Com o controle de cruzeiro acionado a 50 km/h, o condutor fez uso do pedal de aceleração para elevar a velocidade de 50 para 90 km/h, simulando uma possível ultrapassagem. Pelo gráfico do pedal de aceleração, medido em porcentagem, nota-se que o seu acionamento ocorreu em torno de 10 segundos, sendo então liberado em 30 segundos, instante em que o controlador de cruzeiro voltou a atuar no sistema para restabelecer a velocidade com o último valor de referência, no caso 50 km/h. Além disto, observa-se que, após o pedal de aceleração ser liberado, a velocidade atingiu novamente o seu valor de referência em torno do instante 47 segundos, o que valida a funcionalidade do controle de cruzeiro de permitir o controle da aceleração pelo motorista.

Figura 132 – Uso do pedal de aceleração para ultrapassagem com o controle de cruzeiro acionado a 50 km/h.

Fonte: Autor.

4.3.4 Aplicação de degrau na referência do controle de cruzeiro

Com intuito de estabelecer uma comparação com os resultados das simulações da [subseção 4.2.4](#), esta subseção aborda os resultados práticos obtidos por meio da aplicação de degrau na referência do controle de cruzeiro. Para isto, os mesmos três ensaios realizados em ambiente de simulação foram novamente refeitos com o controle de cruzeiro aplicado ao veículo. É importante mencionar que, para a realização destes testes, a lógica no *firmware* que determina a referência do cruzeiro sofreu uma pequena alteração temporária para permitir aplicação de degrau, em vez do valor constante dado pela velocidade no instante do acionamento do cruzeiro.

As Figuras [133](#), [134](#) e [135](#) ilustram, respectivamente, os resultados obtidos com aplicação de degrau 40-50 km/h, 50-80 km/h e 50-120 km/h na referência do controle de cruzeiro. Além disto, estas figuras também registram os resultados obtidos nas simulações executadas com os respectivos degraus.

Figura 133 – Controle de cruzeiro com aplicação de degrau na referência de 40 a 50 km/h.

Fonte: Autor.

Figura 134 – Controle de cruzeiro com aplicação de degrau na referência de 50 a 80 km/h.

Fonte: Autor.

Figura 135 – Controle de cruzeiro com aplicação de degrau na referência de 50 a 120 km/h.

Fonte: Autor.

Com base nas curvas obtidas, é possível concluir que, de modo geral, o controle de cruzeiro implementado no *firmware* da ECU v2.0 apresenta resultados muito próximos daqueles obtidos nas simulações, inclusive no último teste em que se aplicou um degrau de 50 a 120 km/h na referência do sistema.

É importante observar que a presença de um pequeno sobressinal no segundo teste, realizado com degrau 50-80 km/h (Figura 134), não prejudica a aplicação do controle de cruzeiro, em virtude da pequena amplitude deste sobressinal (inferior a 2 km/h). Além disto, a diferença entre os valores de regime estacionário do sinal de controle não representa um problema para o controle de cruzeiro, visto que as curvas de saída da simulação e da planta estão consideravelmente próximas em todos os casos analisados.

4.4 Considerações finais

O presente capítulo abordou as tarefas realizadas na segunda fase do trabalho, responsáveis pelo desenvolvimento de um controlador de cruzeiro para operação no automóvel Polo Sedan 2.0L 2004, o qual não conta com este recurso na sua versão de fábrica. A primeira tarefa consistiu na obtenção de um modelo empírico para o sistema composto pelo veículo equipado com a ECU deste trabalho. Em seguida, de posse de uma função de transferência para a planta, o trabalho realizou o projeto do controlador na ferramenta MATLAB, validando-o posteriormente em ambiente de simulação. Uma vez verificado que as especificações definidas para a aplicação de controle foram devidamente atendidas nas simulações, o controlador foi implementado no *firmware* da ECU v2.0 para, em seguida, ser validado por meio de testes utilizando o automóvel em um dinamômetro inercial.

Os excelentes resultados obtidos nesta segunda fase do trabalho, sobretudo na parte prática envolvendo testes com o veículo, demonstraram que o controle de cruzeiro, mesmo projetado com auxílio de um simples modelo de primeira ordem obtido por meio de experimentação, é capaz de controlar a velocidade do veículo de maneira robusta e eficiente, mantendo o erro de regime limitado a um valor suficientemente pequeno, inclusive na presença de perturbação externa, representada nos testes pela carga do dinamômetro. Além disto, o trabalho introduziu uma importante funcionalidade na malha de controle para permitir que o condutor faça uso do pedal de aceleração enquanto o cruzeiro estiver acionado, de tal modo que o sistema retoma automaticamente o controle da velocidade a partir do momento em que o condutor libera o pedal de aceleração.

5 Conclusões

O projeto Otto IV cumpriu com os seus objetivos iniciais de aprimorar o funcionamento da ECU desenvolvida em 2013 no projeto Otto II (PEREIRA, 2013) e de desenvolver um controlador de cruzeiro para operar em um veículo que não possui este recurso em seu estado de fábrica. Para isto, o trabalho, em termos de organização, foi dividido em duas fases, sendo que a primeira foi responsável pelo aprimoramento da ECU, enquanto a segunda teve como objetivo o desenvolvimento do controle de cruzeiro.

A primeira fase do projeto envolveu diversas tarefas que buscaram aperfeiçoar o funcionamento da ECU de 2013 (denominada no trabalho por ECU v2.0), visando a sua evolução no que diz respeito ao estado da arte de unidades de gerenciamento eletrônico de motores. Para isto, parte das atividades executadas nesta fase foram destinadas a melhorar diversas malhas de controle já existentes no *firmware* desenvolvido em 2013 (a exemplo do controle de marcha lenta), enquanto outras atividades se concentraram na implementação de novos recursos, tais como controle de torque e controle da mistura ar/combustível em malha fechada. Além destas atividades envolvendo desenvolvimento de *firmware*, o trabalho, em sua primeira fase, também promoveu o desenvolvimento de uma nova ferramenta de *software* que, além de ser capaz de monitorar a ECU em tempo real, integra diversas funções, tais como função de computador de bordo alternativo, opção para controle do motor através da simulação do pedal de aceleração, opção para alteração da rotação de marcha lenta e função para automatização do ensaio de identificação do veículo (tarefa essencial para o projeto do controlador de cruzeiro).

A segunda fase do trabalho contemplou o projeto do controlador de cruzeiro, recurso responsável por manter, de maneira autônoma, a velocidade do veículo em torno de um certo valor de referência, sem a necessidade de intervenção do condutor no pedal de aceleração. A tarefa inicial desta segunda fase consistiu na obtenção de um modelo empírico para o sistema composto pelo veículo equipado com a ECU deste trabalho. Em seguida, de posse de uma função de transferência para a planta, o trabalho realizou o projeto do controlador na ferramenta MATLAB, validando-o posteriormente em ambiente de simulação com auxílio da ferramenta Simulink. Uma vez constatado que as especificações de controle foram respeitadas nas simulações, o controlador foi implementado no *firmware* da ECU v2.0 e, em seguida, foi validado por meio de testes utilizando o automóvel em um dinamômetro inercial.

Os excelentes resultados obtidos na primeira fase do trabalho foram decisivos para melhorar consideravelmente o funcionamento do motor, de tal modo que agora o seu comportamento está bem próximo daquele obtido quando se utiliza a sua ECU original

de fábrica. Com isto, o projeto Otto IV cumpre a sua meta inicial de disponibilizar uma ECU didática para uso em futuros projetos na área de eletrônica automotiva, de tal forma que estes passam a ter a disposição uma ECU robusta e estável para ser utilizada como plataforma de trabalho. Além disto, esta primeira fase, ao melhorar determinadas características do sistema como, por exemplo, a dirigibilidade do veículo, contribuiu para o sucesso do projeto do controle de cruzeiro.

Já os excelentes resultados obtidos na segunda fase, sobretudo na parte prática envolvendo testes com o veículo, demonstraram que o controle de cruzeiro, mesmo projetado com auxílio de um simples modelo de primeira ordem obtido por meio de experimentação, é capaz de controlar a velocidade do veículo de maneira robusta e eficiente, mantendo o erro de regime limitado a um valor suficientemente pequeno, inclusive na presença de perturbação externa, representada nos testes pela carga do dinamômetro. Além disto, adicionou-se uma importante funcionalidade na malha de controle para permitir que o condutor faça uso do pedal de aceleração enquanto o cruzeiro estiver acionado, de tal modo que o sistema retoma automaticamente o controle da velocidade a partir do momento em que o condutor libera o pedal de aceleração.

Da mesma forma que ocorreu em 2013 no âmbito do projeto Otto II, o presente trabalho se mostrou bastante desafiador, pois se trata de uma aplicação real de grande porte que envolve diversos conceitos como eletrônica embarcada, teoria de controle, modelagem de sistemas dinâmicos, programação, engenharia automotiva, dentre outros. Assim, além de estimular a busca por novos conhecimentos, o projeto se constituiu em uma grande oportunidade de aplicação de parte dos conceitos adquiridos ao longo dos cursos de graduação e pós-graduação em sistemas eletrônicos da Escola Politécnica da USP.

Espera-se, com o resultados obtidos neste projeto, que se desperte um interesse maior não somente de alunos e professores, mas também de profissionais e empresas do setor automobilístico, em participar e trabalhar, conjuntamente ao departamento PSI da Escola Politécnica da USP e à FATEC Santo André, em novos projetos inovadores na área de eletrônica automotiva. Alunos que buscam desafios em projetos práticos, com o objetivo de aprimorar seus conhecimentos em eletrônica e controle, certamente terão esta oportunidade em um projeto deste tipo, envolvendo atividades como desenvolvimento de *hardware*, *firmware* e *software*.

5.1 Sugestões para projetos futuros

- Desenvolvimento de modelos matemáticos para aplicação da metodologia de projeto por modelos, visando o desenvolvimento de futuros projetos em diferentes motores;
- Modelagem e desenvolvimento de controle de torque em malha fechada, o que envolve estratégia para estimar o seu valor com base em sensores presentes no motor;
- Desenvolvimento de controle de avanço de ignição em malha fechada, o que envolve estratégia para se retirar informação do sensor de detonação¹;
- Desenvolvimento de ECU para motores bicom bustíveis (flex), com uso de sensores de composição de combustível para aprimorar a identificação da mistura do combustível e possibilitar um ajuste eficiente dos mapas de injeção e ignição;
- Desenvolvimento de ECU para controle de transmissão, interagindo diretamente com a ECU v2.0 aprimorada neste projeto;
- Desenvolvimento de estratégia para minimizar o consumo de combustível e a emissão de poluentes;
- Desenvolvimento de controle de cruzeiro adaptativo (*Adaptive Cruise Control*), tomando por base o controle de cruzeiro desenvolvido neste trabalho.
- Implementação de novos recursos de assistência ao motorista (ADAS), interagindo diretamente com a ECU v2.0 aprimorada neste projeto.

¹ Apesar de estar presente no motor Volkswagen 2.0L, este sensor, conhecido também por sensor *knock*, não foi utilizado no presente projeto.

Referências

ANGÉLICO, B. A. *Controle Digital de Sistemas Dinâmicos*. 2016. Apostila para disciplina de pós graduação PTC-5611 do Departamento de Engenharia de Telecomunicações e Controle da Escola Politécnica da USP.

ASHOK, B.; ASHOK, S. D.; KUMAR, C. R. A review on control system architecture of a {SI} engine management system. *Annual Reviews in Control*, v. 41, p. 94 – 118, 2016. ISSN 1367-5788. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S1367578816300086>>.

AUDI AG. *Audi A4 pre sense city*. 2016. Disponível em: <<http://www.audi-technology-portal.de/en/electrics-electronics/driver-assistant-systems/audi-a4-pe-sense-city-en>>. Acesso em: janeiro/2017.

AUDI AG. *Audi active lane assist*. 2016. Disponível em: <http://www.audi-technology-portal.de/en/electrics-electronics/driver-assistant-systems/audi-active-lane-assist_en>. Acesso em: agosto/2016.

AUDI AG. *Audi pre sense*. 2016. Disponível em: <http://www.audi-technology-portal.de/en/electrics-electronics/safety-systems/audi-pre-sense_en>. Acesso em: janeiro/2017.

B&B ELECTRONICS. *OBD-II Background Information*. 2011. Disponível em: <<http://www.obdii.com/background.html>>. Acesso em: agosto/2016.

BELLINI, J. *Aplicativo ‘traduz’ dados de veículo em tempo real para motoristas*. 2014. Disponível em: <<http://g1.globo.com/sao-paulo/sorocaba-jundiai/noticia/2014/10/aplicativo-traduz-dados-de-veiculo-em-tempo-real-para-usuarios.html>>. Acesso em: agosto/2016.

BOSCH, R. *Manual de tecnologia automotiva*. [S.l.]: Edgard Blücher, 2005. ISBN 8521203780.

CALIFORNIA AIR RESOURCES BOARD. *On-Board Diagnostic II (OBD II) Systems - Fact Sheet / FAQs*. 2015. Disponível em: <<http://www.arb.ca.gov/msprog/obdprog/obdfaq.htm>>. Acesso em: agosto/2016.

CAN BUS KIT. *What is CAN Bus?* 2016. Disponível em: <<http://canbuskit.com/what.php>>. Acesso em: agosto/2016.

CARBON CLEANING USA. *What is the pre ignition?* 2016. Disponível em: <<http://www.carboncleaningusa.com/pre-ignition/gas/nissan/gt-r-3-8-1>>. Acesso em: agosto/2016.

COMPANHIA DE ENGENHARIA DE TRÁFEGO. *CET - Dúvidas mais frequentes*. 2017. Disponível em: <http://cetsp1.cetsp.com.br/falecomsac/faq_geral.asp>. Acesso em: março/2017.

COOK, G. *Catalogue of parametrised CRC algorithms with 16 bits*. 2016. Disponível em: <<http://reveng.sourceforge.net/crc-catalogue/16.htm>>. Acesso em: outubro/2016.

- DEUR, J. et al. An electronic throttle control strategy including compensation of friction and limp-home effects. *IEEE Transactions on Industry Applications*, v. 40, n. 3, p. 821–834, 2004. ISSN 0093-9994.
- DIAS, A. *Injeção Eletrônica: Controle Automatizado da Marcha-Lenta*. 2012. Disponível em: <<http://www.carrosinfoco.com.br/carros/2012/07/injecao-eletronica-controle-automatizado-da-marcha-lenta/>>. Acesso em: agosto/2016.
- DONZELLINI, G.; PONTA, D. *Auxiliary Trigger*. 2016. Disponível em: <http://www.esng.dibe.unige.it/deeds/LearningMaterials/LM/T120/120074_TimerDebounce_Int/Index.htm>. Acesso em: novembro/2016.
- FELICE, B. *Implementing The CCITT Cyclical Redundancy Check*. 2007. Disponível em: <<http://www.drdoobs.com/implementing-the-ccitt-cyclical-redundan/199904926>>. Acesso em: outubro/2016.
- FREESCALE. *Datasheet da família MC9S12XE*. [S.l.], 2012.
- FREESCALE. *S12XE: 16-Bit Automotive Microcontroller - Overview*. 2013. Disponível em: <http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=S12XE&nodeId=0162468636K100>. Acesso em: agosto/2016.
- GARCIA, C. *Modelagem e Simulação de Processos Industriais e de Sistemas Eletromecânicos Vol. 1*. EDUSP, 2005. ISBN 9788531409042. Disponível em: <<https://books.google.com.br/books?id=PdxwQs6P0nIC>>.
- GARCIA, C. *Identificação de Sistemas*. 2015. Apostila para disciplina de pós graduação PTC-5719 do Departamento de Engenharia de Telecomunicações e Controle da Escola Politécnica da USP.
- GUZZELLA, L.; ONDER, C. *Introduction to Modeling and Control of Internal Combustion Engine Systems*. Springer, 2004. ISBN 9783540222743. Disponível em: <<https://books.google.com.br/books?id=6QzSnUSP3VoC>>.
- IAN J. HAWKINS. *Torque - OBD2 Performance and Diagnostics for your Vehicle*. 2016. Disponível em: <<https://torque-bhp.com/>>. Acesso em: agosto/2016.
- ISERMANN, R.; MÜNCHHOF, M. *Identification of Dynamic Systems: An Introduction with Applications*. Springer Berlin Heidelberg, 2010. (Advanced Textbooks in Control and Signal Processing Series). ISBN 9783540788799. Disponível em: <<https://books.google.com.br/books?id=tTwYA6ph9LOC>>.
- KIENCKE, U.; NIELSEN, L. *Automotive Control Systems*. [S.l.]: Springer Berlin Heidelberg, 2005. ISBN 9783540231394.
- KILIÇ, . et al. Intelligent adaptive cruise control system design and implementation. In: SYSTEM OF SYSTEMS ENGINEERING CONFERENCE, 10., 2015, San Antonio. **Proceedings...** [S.l.]: IEEE, 2015. p. 232–237.
- LAMMERT BIES. *On-line CRC calculation and free library*. 2015. Disponível em: <<https://www.lammertbies.nl/comm/info/crc-calculation.html>>. Acesso em: outubro/2016.

- LJUNG, L. *System Identification: Theory for the User*. Pearson Education, 1998. ISBN 9780132440530. Disponível em: <<https://books.google.com.br/books?id=fYSrk4wDKPsC>>.
- MANAVELLA, J. H. *Controle integrado do motor*. [S.l.: s.n.], 1996.
- MATHWORKS. *Loss Function and Model Quality Metrics*. 2016. Disponível em: <<https://www.mathworks.com/help/ident/ug/model-quality-metrics.html>>. Acesso em: dezembro/2016.
- MATHWORKS. *What Is a MATLAB Function Block?* 2017. Disponível em: <<https://www.mathworks.com/help/simulink/ug/what-is-a-matlab-function-block.html>>. Acesso em: março/2017.
- MAZDA MOTOR. *LDWS (Lane Departure Warning System)*. 2016. Disponível em: <http://www.mazda.com/en/innovation/technology/safety/active_safety/ldws/>. Acesso em: agosto/2016.
- MICROSOFT. *About Multimedia Timers*. 2016. Disponível em: <[https://msdn.microsoft.com/en-us/library/vs/alm/dd742877\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/vs/alm/dd742877(v=vs.85).aspx)>. Acesso em: novembro/2016.
- MICROSOFT. *Stopwatch Class*. 2016. Disponível em: <[https://msdn.microsoft.com/pt-br/library/system.diagnostics.stopwatch\(v=vs.90\).aspx](https://msdn.microsoft.com/pt-br/library/system.diagnostics.stopwatch(v=vs.90).aspx)>. Acesso em: novembro/2016.
- NISSAN MOTOR. *Driver Attention Alert System: A safety feature that helps reduce drowsy driving*. 2015. Disponível em: <<http://www.nissanusa.com/blog/driver-attention-alert>>. Acesso em: agosto/2016.
- NORTON, J. *An Introduction to Identification*. Dover Publications, 2009. (Dover Books on Electrical Engineering Series). ISBN 9780486469355. Disponível em: <https://books.google.com.br/books?id=eyHC7751n_cC>.
- NVIDIA. *Advanced Driver Assistance Systems (ADAS)*. 2016. Disponível em: <<http://www.nvidia.com/object/advanced-driver-assistance-systems.html>>. Acesso em: agosto/2016.
- OBD-CODES.COM. *OBD-II (Check Engine Light) Trouble Codes*. 2016. Disponível em: <http://www.obd-codes.com/trouble_codes/>. Acesso em: agosto/2016.
- OBSERVATÓRIO NACIONAL DE SEGURANÇA VIÁRIA. *Sono e cansaço são responsáveis por até 50 % dos acidentes em rodovias*. 2015. Disponível em: <<http://www.onsv.org.br/noticias/sono-e-cansaco-sao-responsaveis-por-ate-50-dos-acidentes-em-rodovias/>>. Acesso em: agosto/2016.
- OGATA, K. *Engenharia de controle moderno*. Prentice Hall, 2003. ISBN 9788587918239. Disponível em: <<http://books.google.com.br/books?id=fXmIPQAACAAJ>>. Acesso em: outubro/2016.
- ORGANIZAÇÃO MUNDIAL DA SAÚDE. *OMS: Brasil é o país com maior número de mortes de trânsito por habitante da América do Sul*. 2015. Disponível em: <<https://nacoesunidas.org/oms-brasil-e-o-pais-com-maior-numero-de-mortes-de-transito-por-habitante-da-america-do-sul/>>. Acesso em: agosto/2016.

- OSMAN, K.; RAHMAT, M. F.; AHMAD, M. A. Modelling and controller design for a cruise control system. In: INTERNATIONAL COLLOQUIUM ON SIGNAL PROCESSING & ITS APPLICATIONS, 5., 2009, Kuala Lumpur. **Proceedings...** [S.l.]: IEEE, 2009. p. 254–258.
- PEREIRA, B. C. F. *Unidade de Gerenciamento Eletrônico de um Motor Volkswagen 2.0L: Projeto Otto II*. Monografia (Trabalho de Conclusão de Curso) — Escola Politécnica da Universidade de São Paulo, 2013.
- PEREIRA, B. S. *Controle da Mistura Ar/Combustível em um Motor a Combustão Interna: Sistema em Malha Fechada*. Dissertação (Mestrado) — Escola Politécnica da Universidade de São Paulo, 2015.
- PORTAL VRUM. *Ficha Técnica: Polo Sed./ Sed. COMFORT. 2.0 Mi 8V 4p*. 2013. Disponível em: <<http://www.vrum.com.br/fichatecnica/Volkswagen/POLO/2004/005186-1>>. Acesso em: dezembro/2016.
- PSA GROUP. *Fully automatic parking, a reality from 2017*. 2016. Disponível em: <<http://www.psa-peugeot-citroen.com/en/featured-content/connected-car/city-park-full-automatic>>. Acesso em: agosto/2016.
- ROBERT BOSCH GMBH. *CAN Specification Version 2.0*. [S.l.], 1991. Disponível em: <<http://www.kvaser.com/software/7330130980914/V1/can2spec.pdf>>. Acesso em: agosto/2016.
- ROBERT BOSCH GMBH. *Sensors*. [S.l.], 2013.
- ROBERT BOSCH GMBH. *Electronic Engine Management: Perfectly-Regulated Engine Functions*. 2017. Disponível em: <http://de.bosch-automotive.com/en/parts_and_accessories/motor_and_sytems/benzin/engine_management/engine_management_4>. Acesso em: Janeiro/2017.
- SANTOS, J. P. F. D.; ROSSETTI, P. D. C. *Otto III by FlexECU - Gerenciamento Eletrônico de um Motor VW 1.6L*. Monografia (Trabalho de Conclusão de Curso) — Escola Politécnica da Universidade de São Paulo, 2015.
- SCARPINETTI, V. S.; SOARES, A. M. F. *Unidade eletrônica de controle de um motor a combustão: Projeto Otto*. Monografia (Trabalho de Conclusão de Curso) — Escola Politécnica da Universidade de São Paulo, 2012.
- SHAOUT, A.; COLELLA, D.; AWAD, S. Advanced driver assistance systems - past, present and future. In: INTERNATIONAL COMPUTER ENGINEERING CONFERENCE, 7., 2011, Giza. **Proceedings...** [S.l.]: IEEE, 2011. p. 72–82.
- SHEDEED, M. et al. Functional design and verification of automotive embedded software: An integrated system verification flow. In: SAUDI INTERNATIONAL ELECTRONICS, COMMUNICATIONS AND PHOTONICS CONFERENCE, 2013, Fira. **Proceedings...** [S.l.]: IEEE, 2013. p. 1–5.
- TOYOTA MOTOR AUSTRALIA. *Cruise Control. For less tiring driving*. 2016. Disponível em: <<http://www3.toyota.com.au/yaris/features/technology/cruise-control>>. Acesso em: agosto/2016.

- VIJAY, E. V. et al. Electronic control unit for an adaptive cruise control system and engine management system in a vehicle using electronic fuel injection. In: INTERNATIONAL CONFERENCE ON EMERGING TRENDS IN ROBOTICS AND COMMUNICATION TECHNOLOGIES, 8., 2010, Chennai. **Proceedings...** [S.l.]: IEEE, 2010. p. 143–146.
- VOLKSWAGEN. *Adaptive Cruise Control (ACC): Volkswagen's safe distance technology*. 2016. Disponível em: <<http://www.volkswagen.co.uk/technology/adaptive-cruise-control-acc>>. Acesso em: agosto/2016.
- VOLKSWAGEN. *Cruise control: Automatically keeps your car moving at a safe, constant speed*. 2016. Disponível em: <<http://www.volkswagen.co.uk/technology/driving/cruise-control>>. Acesso em: agosto/2016.
- VOLKSWAGEN. *Park assist: Helps you park your car quickly and easily*. 2016. Disponível em: <<http://www.volkswagen.co.uk/technology/parking-and-manoeuvring/park-assist>>. Acesso em: agosto/2016.
- VOLKSWAGEN. *Warning lights: Warning lights and what they mean*. 2016. Disponível em: <<http://www.volkswagen.co.uk/owners/warning-lights>>. Acesso em: agosto/2016.
- VOLKSWAGEN OF THE WOODLANDS. *How to use Volkswagen Touareg adaptive cruise control*. 2015. Disponível em: <<http://www.vwofthewoodlands.com/blog/how-to-use-volkswagen-touareg-adaptive-cruise-control/>>. Acesso em: agosto/2016.
- XIONG, Q. et al. Research on idle speed control for efi gasoline engine with etb. In: IEEE INTERNATIONAL CONFERENCE ON COGNITIVE INFORMATICS, 8., 2009, Kowloon. **Proceedings...** [S.l.]: IEEE, 2009. p. 222–226.
- YE, Z. Modeling, identification, design, and implementation of nonlinear automotive idle speed control systems - an overview. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, v. 37, n. 6, p. 1137–1151, 2007. ISSN 1094-6977.
- ZHU, D. et al. Idle speed control system design based on engine torque management. In: CHINESE AUTOMATION CONGRESS, 8., 2015, Wuhan. **Proceedings...** [S.l.]: IEEE, 2015. p. 1809–1814.

Apêndices

APÊNDICE A – *Firmware*

A seguir foram disponibilizados trechos do *firmware* dos blocos que integram a ECU deste trabalho. Vale ressaltar que cada bloco é comandado por um microcontrolador automotivo cujo código embarcado foi desenvolvido em linguagem C.

A.1 Bloco de Gerenciamento

A.1.1 Controle de Cruzeiro

```
// Rotina do controle de cruzeiro (controlador Dahlin)
// OBS: Deve ser chamada a cada 100ms!
#define N_CRUZEIRO 7 // Numero de amostras utilizadas na malha de
    controle
void controle_cruzeiro() {
    static const float Max_Control = MAX_REF_TORQUE; // Limitacao
        superior da ref. de torque
    static const float Min_Control = MIN_REF_TORQUE; // Limitacao
        inferior da ref. de torque
    float erro_vel = 0; // Erro
    float min_init_control = 0; // Sinal de controle inicial
        minimo (p/ evitar queda na velocidade quando o cruzeiro for
        ligado)
    static float erro_vel_ant = 0; // Erro anterior
    static float control_ant[N_CRUZEIRO]; // Sinal de controle
        anterior (posição "0" = valor atual)
    static unsigned char modo_cruzeiro_ant = 0; // Estado do
        cruzeiro no laço anterior
    static unsigned char flRetomada = 0;
    unsigned int i = 0; // contador

    // Verifica qualquer mudança no estado do acionamento do
        cruzeiro
    if (modo_cruzeiro != modo_cruzeiro_ant) {
        if (modo_cruzeiro) { // Cruzeiro acabou de ser ativado?

            // Atualiza leitura do pedal
            pedal1_check();
```

```
#ifdef STEP_REFCRUISE // Cruzeiro com referencia fixa
    (ensaio em degrau)?
    ref_velocidade = STEP_REFCRUISE_VINIT; // Define a
        ref do cрузeiro com o valor inicial do degrau
    min_init_control = pedal; // Assume sinal de controle
        minimo somente com o valor do pedal
    timerRefCruzeiro = STEP_REFCRUISE_STEPTIME; // Ativa
        timer para aplicação de degrau na ref. do cрузeiro
#else
    // Referencia de velocidade
    ref_velocidade = (unsigned char)velocidade_medida;

    // Determinacao do sinal de controle inicial minimo
    if (ref_velocidade < 40) // 0 a 39km/h
        min_init_control = 10.0;
    else if (ref_velocidade < 70) // 40 a 69km/h
        min_init_control = 20.0;
    else // 70km/h a infinito!
        min_init_control = 25.0;
#endif

    // Anula valor anterior do erro
    erro_vel_ant = 0;

    // Injeta sinal de controle para "segurar" a
        velocidade
    for (i = 0; i < N_CRUZEIRO; i++)
    {
        control_ant[i] = pedal;
        if (control_ant[i] < min_init_control)
            control_ant[i] = min_init_control; // Garante
                um sinal de controle minimo para a
                velocidade nao cair muito!
    }

    // Atraso para liberar o pedal ao motorista, caso
        seja necessario
    timerCruzeiro = DELAY_CRUISE_PEDAL;
    flRetomada = 0;
}

else { // Cruzeiro acabou de ser desligado!
    ref_velocidade = 0;
```

```
    }  
}  
  
// Calculo do erro  
erro_vel = (float)ref_velocidade - velocidade_medida;  
  
// Calculo do sinal de controle  
if (flRetomada) // Motorista controlando?  
    control_ant[0] = control_ant[1]; // Mantem o ultimo sinal  
    de controle!  
else  
    control_ant[0] = 0.8825 * control_ant[1] + 0.1175 *  
        control_ant[6] + 2.357 * erro_vel - 2.31 * erro_vel_ant  
        ; // C(z) - Controlador Dahlin  
  
// Limitacao da acao de controle (ref torque)  
if (control_ant[0] > Max_Control)  
    control_ant[0] = Max_Control;  
else if (control_ant[0] < Min_Control)  
    control_ant[0] = Min_Control;  
  
// Desloca amostras para o proximo laço de controle  
erro_vel_ant = erro_vel;  
modo_cruzeiro_ant = modo_cruzeiro;  
for (i = N_CRUZEIRO-1; i > 0; i--)  
    control_ant[i] = control_ant[i-1];  
  
// Verifica se o condutor acionou o pedal do acelerador (Ex.:  
    Ultrapassagem!)  
if (timerCruzeiro == 0) {  
    pedal1_check();  
    if (pedal > control_ant[0]) {  
        control_ant[0] = pedal; // Passa o controle do  
            acelerador para o condutor temporariamente!  
        flRetomada = 1;  
    }  
    else {  
        flRetomada = 0;  
    }  
}  
}
```

```
    ref_torque_cruzeiro = control_ant[0]; // Passa o sinal de
        controle p/ a variavel final!
}

// Rotina que verifica condicoes para abortar o modo de cruzeiro!
void verifica_cruzeiro() {
    if ((freio != 0)                                // Freio acionado?
        || (embreagem != 0)                        // ou embreagem acionada?
        #ifndef STEP_REFCRUISE
        || (marcha_engatada < 2)                    // ou veiculo desengatado
            ou em 1ª marcha?
        #endif
        || (velocidade_medida < 10.0)                // ou velocidade abaixo
            de 10 km/h?
        || (erro_hw != ERRO_HW_OK))                 // ou erro de hardware?
        modo_cruzeiro = 0; // Desativa o cruzeiro!!
}
```

A.1.2 Gerenciamento entre controle de torque e controle de marcha lenta

```
// Rotina de gerenciamento entre controle de torque e controle de
// marcha lenta
// OBS: Chamada a cada 10ms!
void controle_rpm_torque() {
    float control_torque, control_mlenta;
    float ref_vb_temp;
    float ref_vb_max, ref_vb_min;

    control_mlenta = controle_rpm_mlenta();
    control_torque = controle_torque();

    // Sinal de controle resultante = ref de posicao da VB
    if (ref_torque < REF_TORQUE_MLENTA) // Marcha lenta?
        ref_vb_temp = control_mlenta;
    else // Controle de torque!
        ref_vb_temp = control_torque;

    // Calculo da limitacao da posicao max da vb para o motor nao
    // "afogar" em aceleracoes grandes
    if (rotacao < 700)
        ref_vb_max = TPS_MAX_1;
    else if (rotacao < 1700)
        ref_vb_max = TPS_MAX_2;
    else if (rotacao < 3000)
        ref_vb_max = TPS_MAX_2 + ((TPS_MAX - TPS_MAX_2) * ((float)
            rotacao - 1700.0) / (3000.0 - 1700.0));
    else if (rotacao < 5000)
        ref_vb_max = TPS_MAX;
    else { // rotacao > 5000 RPM
        ref_vb_max = TPS_MAX + ((TPS_MAX_6000 - TPS_MAX) * ((
            float)rotacao - 5000.0) / (6000.0 - 5000.0));
        if (ref_vb_max < TPS_MAX_6000)
            ref_vb_max = TPS_MAX_6000;
    }

    // Calculo da limitacao da posicao min da vb
    ref_vb_min = TPS_LOWER_LIMIT; // Impede fechamento total da
    vb

    // Limita referencia para controle da borboleta
```

```
    if (ref_vb_temp < ref_vb_min)
        ref_vb_temp = ref_vb_min;
    else if (ref_vb_temp > ref_vb_max) // Limitacao para o motor
        nao "afogar" em aceleracoes grandes
        ref_vb_temp = ref_vb_max;

    ref_vb = filtroRefVB(ref_vb_temp); // Filtro da ref. de
        posicao da VB
}
```


A.1.3 Controle de torque em malha aberta

```
// Rotina para controle de torque em malha aberta (estimacao)
float controle_torque() {
    float control = 0;
    float abertura_vb = 0; // Referencia de abertura da VB (0 a
        100%)
    float abertura_vb_max = 1.0; // Maxima abertura da VB (em %)

    if (modo_operacao == MODO_OPERACAO_ECONOMICO) // Modo de
        operacao economico?
        abertura_vb_max = 0.5; // 50% de abertura maxima

    // Calculo da abertura da VB (0 até "abertura_vb_max")
    // Baseia-se na curva  $y = 0.01 \cdot x^2$  (parabola)
    //
    //          TPS_MAX * abertura_vb_max -
    //                               - :
    //                               - :
    //                               - :
    //                               - :
    //                               - :
    //                               - :
    //          tps_base_mlenta -      :
    //                               :   :
    //                               :   :
    // Ref. Torque ->    0%           100%
    //
    //
    abertura_vb = 0.01 * ref_torque * ref_torque *
        abertura_vb_max;
    if (abertura_vb < 0)
        abertura_vb = 0;
    else if (abertura_vb > 100.0)
        abertura_vb = 100.0;

    // Converte a abertura da VB em valor de TPS
    control = tps_base_mlenta + (TPS_MAX - tps_base_mlenta) *
        abertura_vb * 0.01;

    return control;
}
```

A.1.4 Controle de marcha lenta por atuação na borboleta eletrônica

```
// Rotina para controle da rotacao de marcha lenta por atuacao na
VB (Controle PI)
// OBS: Chamada a cada 10ms!
float controle_rpm_mlenta() {
    static const float Kp_rpm_ign = 0.05; // Ganho Kp quando a
        IGN atua na marcha lenta
    static const float Ki_rpm_ign = 0.1; // Ganho Ki quando a IGN
        atua na marcha lenta
    static const float Kp_rpm_no_ign = 0.005; // Ganho Kp quando
        a IGN nao atua na marcha lenta
    static const float Ki_rpm_no_ign = 0.015; // Ganho Ki quando
        a IGN nao atua na marcha lenta

    #ifdef COMPENSAR_IGN_MLENTA // Correcao na ign para marcha
        lenta esta ativada?
    // Ganhos podem ser maiores neste caso!
    float Kp_rpm = Kp_rpm_ign;
    float Ki_rpm = Ki_rpm_ign;
    #else // Sem correcao na ign...
    // Sem ajuda da ign, os ganhos devem ser menores para nao
        deixar o controle instavel!
    float Kp_rpm = Kp_rpm_no_ign;
    float Ki_rpm = Ki_rpm_no_ign;
    #endif

    static const float Ts = 0.01; // Período do controle = 10ms
    static const float Max_Control = TPS_LIMP_HOME; // Limitacao
        superior da ref. da VB
    float Min_Control = TPS_LOWER_LIMIT; // Limitacao inferior da
        ref. da VB
    float erro_rpm; // Erro
    float control_P, control_I; // Acao proporcional e integral
        do controlador
    float control = tps_base_mlenta; // Sinal de controle
    float control_step_vb = 0;
    unsigned char flSaturacao = 0; // Se igual a 1 indica que
        houve saturacao do sinal de controle (=ref. VB)
    static float control_ant = TPS_MLENTA; // Sinal de controle
        anterior
    static float control_I_ant = 0; // Acao integral anterior
```

```
#ifndef STEP_VB_MLENTA
// Ensaio de identificacao do modelo da rotacao de marcha
// lenta via degrau no sinal de ref. da VB
control_step_vb = controle_step_vb_m lenta();
if (control_step_vb != 0)
    return control_step_vb;
#endif

// Condições para ativar o controle da rotacao de marcha
// lenta:
// 1 - Pedal solto (ref. de torque "pequena");
// 2 - Rotacao abaixo do limite "rotacao_m lenta_control";
if ((ref_torque < REF_TORQUE_MLENTA) && (rotacao <
    rotacao_m lenta_control)) { // marcha lenta?

    // Calculo do erro
    erro_rpm = (float)rotacao_m lenta - (float)rotacao;

    // Limitacao inferior da ref. da VB deve ser maior quando
    // o veiculo estiver engatado!
    if (marcha_engatada != 0)
        Min_Control = TPS_MLENTA;

    // Em caso de erro de hardware, os ganhos devem ser
    // menores pois a IGN nao atua na marcha lenta neste caso!
    if (erro_hw != ERRO_HW_OK) {
        Kp_rpm = Kp_rpm_no_ign;
        Ki_rpm = Ki_rpm_no_ign;
    }

    // Ganho Kp nao atua se o erro for negativo
    if (erro_rpm < 0)
        Kp_rpm = 0;

    if (((control_ant <= Min_Control) && (erro_rpm < 0)) //
        Houve saturacao do sinal de controle?
        || ((control_ant >= Max_Control) && (erro_rpm > 0)))
        flSaturacao = 1; // Seta a flag de saturacao!

    control_P = Kp_rpm * erro_rpm; // Calcula a acao
    // proporcional do controle
```

```
        if (flSaturacao) // Saturacao?
            control_I = control_I_ant; // Congela a acao
            integral do controle (Anti-Windup)!
        else
            control_I = control_I_ant + Ki_rpm * Ts *
            erro_rpm; // Calcula a acao integral do
            controle

        control += (control_P + control_I); // Sinal de controle
        = P + I

        // Limitacao da acao de controle (ref VB)
        if (control > Max_Control)
            control = Max_Control;
        else if (control < Min_Control)
            control = Min_Control;
    }
    else { // Saiu da marcha lenta...
        control_I = 0; // Zera a acao integral!
    }

    // Guarda os valores para o proximo laco de controle
    control_I_ant = control_I;
    control_ant = control;

    return control;
}
```

A.1.5 Controle de marcha lenta por atuação no avanço de ignição

```
// Rotina para controle da rotacao de marcha lenta por atuacao no
// avanço de IGN (Controle PD)
// OBS: Sinal de controle positivo -> atrasa a IGN; Sinal de
// controle negativo -> adianta a IGN!
float controle_rpm_mlenta_ign() {
    static const float Kp_ign = 0.04; // Ganho P
    static const float Kd_ign = 0.003; // Ganho D
    static const float Max_Control = 2.2; // Limitacao superior
        da correcao do avanço de ign
    static const float Min_Control = -1.5; // Limitacao inferior
        da correcao do avanço de ign
    float erro_rpm = 0; // Erro
    float control_P = 0, control_D = 0; // Acao proporcional e
        derivativa do controlador
    float control = 0; // Sinal de controle
    static float erro_rpm_ant = 0; // Erro anterior

    // Calculo do erro
    erro_rpm = (float)rotacao_mlenta - (float)rotacao;

    // Condições para ativar o controle da rotacao de marcha
    // lenta por avanço de ignicao:
    // 1 - Pedal solto (ref. de torque "pequena");
    // 2 - Rotacao abaixo do limite "rotacao_mlenta_control";
    // 3 - Regime pos-partida
    // 4 - Sem erro de hardware
    if ((ref_torque < REF_TORQUE_MLENTE) && (rotacao <
        rotacao_mlenta_control) && (flPosPartida) && (erro_hw ==
        ERRO_HW_OK)) { // marcha lenta?

        control_P = Kp_ign * erro_rpm; // Calcula a acao
            proporcional do controle
        control_D = Kd_ign * (erro_rpm - erro_rpm_ant); //
            Calcula a acao derivativa do controle

        control = - (control_P + control_D); // Sinal de controle
            = P + D

        // Limitacao da acao de controle
        if (control > Max_Control)
```

```
        control = Max_Control;
    else if (control < Min_Control)
        control = Min_Control;
}
else {
    // Fora do regime de marcha lenta nao se faz correcao na
    // ign!
    control = 0;
}

// Guarda os valores para o proximo laco de controle
erro_rpm_ant = erro_rpm;

return control;
}
```

A.1.6 Cálculo da referência de rotação da marcha lenta

```

// Rotina para determinacao da ref. da rotacao de marcha lenta.
// OBS: Tambem calcula a posicao "base" da VB para este regime.
void calcula_ref_mlenta() {
    unsigned int rotacao_mlenta_temp = MAX_RPM_MLENTE;

    // Calculo da ref. de rotacao de marcha lenta e da posicao "
    // base" da VB em funcao da temperatura do motor
    //
    //
    // MAX_RPM_MLENTE -----
    //                               : -
    //                               :  -
    //                               :   -
    //                               :    -----
    // MIN_RPM_MLENTE
    //                               :      :
    //                               30°C   50°C
    //
    // TPS_MLENTE_FRIO -----
    //                               : -
    //                               :  -
    //                               :   -
    //                               :    -----
    // TPS_MLENTE
    //                               :      :
    //                               30°C   50°C
    //
    if (t_agua < 30.0) { // Motor frio?
        rotacao_mlenta_temp = MAX_RPM_MLENTE;
        tps_base_mlenta = TPS_MLENTE_FRIO;
    }
    else if (t_agua < 50.0) { // Motor semi-frio?
        rotacao_mlenta_temp = (unsigned int)((float)
            MAX_RPM_MLENTE - (((float)MAX_RPM_MLENTE - (float)
            MIN_RPM_MLENTE)*(t_agua - 30.0))*0.05);
        tps_base_mlenta = TPS_MLENTE_FRIO - ((TPS_MLENTE_FRIO -
            TPS_MLENTE)*(t_agua - 30.0)*0.05);
    }
    else {
        // Motor quente!
    }
}

```

```
rotacao_mlenta_temp = MIN_RPM_MLENTA;
tps_base_mlenta = TPS_MLENTA;

// Altera a marcha lenta de acordo com a variavel "
modo_ref_mlenta" definida pela aplicacao!
if (modo_ref_mlenta > 0) {
    rotacao_mlenta_temp = 600 + ((modo_ref_mlenta - 1) *
    50);
    if (rotacao_mlenta_temp > MAX_RPM_MLENTA)
        rotacao_mlenta_temp = MAX_RPM_MLENTA;
}
}

// Marcha lenta passa a ser no minimo 900 RPM se o Ar
Condicionado estiver ligado!
if ((modo_ArCond != 0) && (rotacao_mlenta_temp < 900))
    rotacao_mlenta_temp = 900;

// Marcha lenta com veiculo em movimento e/ou embreagem
pressionada deve ser no minimo 1000 RPM!
// OBS: Melhorar a troca de marcha e auxiliar a arrancada do
veículo!
if (((velocidade > 0) || (embreagem != 0)) && (
rotacao_mlenta_temp < 1000))
    rotacao_mlenta_temp = 1000;

// Condicao pre partida exige uma ref. de marcha lenta maior
para estabilizar o motor!
if (!flPosPartida)
    rotacao_mlenta_temp = MAX_RPM_MLENTA;

// Por segurança, a marcha lenta deve ser no minimo 1000 RPM
em caso de erro de hardware!
if ((erro_hw != ERRO_HW_OK) && (rotacao_mlenta_temp < 1000))
    rotacao_mlenta_temp = 1000;

// Filtra a ref. da marcha lenta para evitar variacoes
bruscas!
rotacao_mlenta = filtroRefMLenta(rotacao_mlenta_temp);

// Calculo da rotacao limite de ativacao o controle de marcha
lenta
```



```
// OBS: Por convencao, o controle de marcha lenta comeca a
    atuar sempre a 300 RPM (ou 500 RPM p/ motor frio) acima da
    sua rotacao de referencia.
// OBS 2: Abaixo deste valor de rotacao nao ocorre corte de
    injecao por desaceleracao!
if (t_agua < 50.0) // Motor frio?
    rotacao_mlenta_control = rotacao_mlenta + 500;
else
    rotacao_mlenta_control = rotacao_mlenta + 300;
}
```

A.1.7 Controle de posição da válvula borboleta eletrônica

```
// Rotina para controle de posicao da VB (controle PID)
// OBS: Chamada a cada 10ms!
void controle_VB() {
    static const float Kp_vb = 250.0; // Ganho P
    static const float Ki1_vb_lp = 3000.0; // Ganho I para fechar
        a VB quando abaixo do Limp Home (Marcha Lenta)
    static const float Ki2_vb_lp = 9000.0; // Ganho I para abrir
        a VB quando abaixo do Limp Home (Marcha Lenta)
    static const float Ki1_vb = 2000.0; // Ganho I para abrir/
        fechar a VB quando acima do Limp Home
    static const float Kd_vb = 500.0; // Ganho D
    static const float Ts = 0.01; // Período do controle = 10ms
    static const float Max_Control = 90000.0; // Limitacao
        superior do PWM (90%)
    static const float Min_Control = -90000.0; // Limitacao
        inferior do PWM (90% "invertido")
    float Ki_vb; // Ganho Ki
    float erro_vb; // Erro
    float control_P, control_I, control_D; // Acao proporcional,
        integral e derivativa do controlador
    float control; // Sinal de controle
    unsigned char flInversao = 0; // Se igual a 1 indica que se
        deve inverter o PWM da VB (sinal de controle negativo)
    unsigned char flSaturacao = 0; // Se igual a 1 indica que
        houve saturacao do sinal de controle (=PWM)
    static float erro_ant = 0; // Erro anterior
    static float control_ant = 0; // Sinal de controle anterior
    static float control_I_ant = 0; // Acao integral anterior

    if ((ref_vb < TPS_MIN) || (flControlarVB == 0)) {
        pwm_33186_control(0, 0); // Desliga a VB!
        return; // Nao Atua na borboleta
    }

    // Calculo do erro
    TPS2_check();
    erro_vb = ref_vb - tps;

    if (((control_ant <= Min_Control) && (erro_vb < 0)) // Houve
        saturacao do sinal de controle?
```

```
|| ((control_ant >= Max_Control) && (erro_vb > 0)))
    flSaturacao = 1; // Seta a flag de saturacao!

// Determinacao do ganho Ki
// Tratamento para evitar que a VB caia abaixo da sua
// referencia na regioa abaixo do Limp Home.
// OBS: Necessario para impedir afogamento do motor durante o
// controle da rotacao de marcha lenta!
if (ref_vb <= TPS_MLENTA) { // VB abaixo do LP?
    if (erro_vb > 0)
        Ki_vb = Ki2_vb_lp; // Abre rapido a VB p/ nao afogar
        a marcha lenta!
    else
        Ki_vb = Ki1_vb_lp;
}
else // VB acima do LP
    Ki_vb = Ki1_vb;

control_P = Kp_vb * erro_vb; // Calcula a acao proporcional
do controle
control_D = Kd_vb * (erro_vb - erro_ant); // Calcula a acao
derivativa do controle

if (flSaturacao) // Saturacao?
    control_I = control_I_ant; // Congela a acao integral do
    controle (Anti-Windup)!
else
    control_I = control_I_ant + Ki_vb * Ts * erro_vb; //
    Calcula a acao integral do controle

control = control_P + control_I + control_D; // Sinal de
controle = P + I + D

// Limitacao da acao de controle (PWM)
if (control > Max_Control)
    control = Max_Control;
else if (control < Min_Control)
    control = Min_Control;

// Guarda os valores p/ o proximo laço de controle
erro_ant = erro_vb;
control_I_ant = control_I;
```

```
control_ant = control;

if (control < 0) {
    control = -control;
    flInversao = 1;
}

// Geracao do PWM para controle da ponte H 33186
pwm_33186_control(control, flInversao);

// Verifica a possibilidade de erro com o mecanismo da VB!
verifica_falha_VB(erro_vb, tps);
}
```

A.1.8 Cálculo do tempo de injeção

```
...

// Valores iniciais dos fatores de correcao
fator_inj = 1.0;
fator_corr_vbat = 1.0;

// Determinacao do fator de correcao dos tempos em funcao da
    tensao da bateria
#ifdef COMPENSAR_VBAT
if (bateria < 14.0) {
    fator_corr_vbat = (7.9 * expf(-0.15 * bateria));
    if (fator_corr_vbat < 1.0)
        fator_corr_vbat = 1.0;
}
#endif

// Calculo da massa de ar admitida por cilindro
// OBS: Utiliza a Lei dos gases ideais ( $PV/4=nRT$ )
m_ar = 0.25 * pressao_ar * 1.984 * 28.96 / (8.314 * t_ar); //
    Unidade: g

// Calculo da massa de combustivel para lambda=1
m_comb = m_ar / 12.7; // Unidade: g

// Correcao empirica da massa de combustivel calculada
// OBS: Necessario para que a estequiometria em malha aberta
    esteja proxima de 1!
m_comb *= 1.04;

// Calculo do tempo de injecao (Fluxo: 176.93 g/min)
tempo_inj_temp = m_comb * 339117.17; // Unidade: us

if ((modo_operacao == 0) && (flPosPartida)) { // Modo de operacao
    "normal" e regime pos partida?
        // Ajuste do ganho de injecao de acordo com a ref de
            torque
        fator_inj += Kp_inj*ref_torque;

        // Ganho para "tirar" o motor do giro de marcha lenta
            (800 RPM) sem afogamento
```

```

        if ((rotacao < 1000) && (ref_torque > REF_TORQUE_MLENTA)
            && (fator_inj < 1.1))
            fator_inj = 1.1;
    }

    // Ganho de injeção p/ partida a frio
    if (!flPosPartida)
        fator_inj += ganho_inj_partida;

    // Fator de correção na injeção para motor frio
    //
    //
    //   +10%  -----
    //                               : -
    //                               :  -
    //                               :   -
    //                               :    -----  0 % (sem
    //      ganho)
    //                               :      :
    //                               30°C   50°C
    //
    if (t_agua < 30.0)
        fator_inj += 0.1;
    else if (t_agua < 50.0)
        fator_inj += (0.25 - 0.005*t_agua);

    // Ajusta o fator de correção de acordo com o sensor da sonda
    //   lambda
    #ifdef CONTROLAR_LAMBDA
    fator_inj += controle_lambda();
    #endif

    // Cálculo da limitação superior do fator de injeção
    //
    //
    //   1.4  -----
    //                               : -
    //                               :  -
    //                               :   -
    //                               :    -----  1.0
    //                               :      :
    //                               5000 RPM  6000 RPM

```

```
//
if (rotacao < 5000)
    fator_inj_max = 1.4; // equivale a lambda ~ 0.71
else if (rotacao < 6000)
    fator_inj_max = 1.4 - 0.0004*((float)rotacao - 5000.0);
else
    fator_inj_max = 1.0;

// Limitacao superior do fator de injecao
if (fator_inj > fator_inj_max)
    fator_inj = fator_inj_max;
// Limitacao inferior do fator de injecao
else if (fator_inj < 0.5)
    fator_inj = 0.5; // equivale a lambda = 2.0

tempo_inj_temp *= fator_inj; // Aplica o fator de correcao
fator_inj_10000 = (unsigned int)(fator_inj * 10000.0); // Guarda
    o fator de inj. para envio ao COM

tempo_inj_temp *= fator_corr_vbat; // Aplica o fator de correcao
    da tensao de bateria

// Limitacao superior do tempo de injecao
if (tempo_inj_temp > 18000.0)
    tempo_inj_temp = 18000.0;

// Limitacao inferior do tempo de injecao
if (tempo_inj_temp < 2500.0)
    tempo_inj_temp = 2500.0;

...
```

A.1.9 Controle da mistura a/c em malha fechada (controle lambda)

```
// Rotina para controle da mistura a/c em malha fechada (controle
lambda - PI)
float controle_lambda() {
    static const float KPlambda = 0.005; // Kp fixo, variação
        pequena
    static const float Max_Control = 0.15; // Limitacao superior
        do incremento de injecao
    static const float Min_Control = -0.15; // Limitacao inferior
        do incremento de injecao
    float KIlambda; // Ganho Integral
    float Ts; // Período do controle
    float erro_lambda = 0; // Erro
    float control = 0; // Sinal de controle = incremento do fator
        multiplicativo do tempo de injecao
    float delta_control_P, delta_control_I; // Incremento da acao
        proporcional e integral do controlador
    unsigned char Rlambda; // 1 - Mistura Rica; 0 - Mistura Pobre
    unsigned char flSaturacao = 0; // Se igual a 1 indica que
        houve saturacao do sinal de controle
    static unsigned char flAceleracao = 0; // Se igual a 1 indica
        que houve uma aceleracao
    static float erro_lambda_ant = 0; // Erro anterior
    static float control_ant = 0; // Sinal de controle anterior

    // Sinal de controle inicia com o valor anterior
    control = control_ant;

    // Verifica condicoes para executar o controle lambda em
        malha fechada:
    // * rotacao entre 500 e 5000 RPM
    // * rotacao constante (|variacao| < 150 RPM/s)
    // * ref. de torque < 70%
    // * Temperatura da agua > 50 graus Celsius (motor quente)
    // * Tempo p/ aquecimento da lambda atingido (
        flLambdaAquecida = 1)
    // * Tensão da lambda menor que ~2,5V (confirma lambda
        aquecida)
    // * Sem erro de hardware
    if ((t_agua > 50.0) && (flLambdaAquecida) && (lambda < 127)
        && (erro_hw == ERRO_HW_OK)) {
```



```
if ((rotacao > 500) && (rotacao < 5000) && (fabsf((float)
delta_rotacao) < MAX_DELTA_RPM_LAMBDA) && (ref_torque <
70.0) && (LambdaTimer == 0)) {
    if (flAceleracao) {
        // Ocorreu uma aceleracao anterior!
        LambdaTimer = 500; // 5s p/ começar a controlar a
            lambda
        flAceleracao = 0; // Reseta flag
        control = 0; // Zera a correcao acumulada!
    }
    else {
        if (lambda < 51) { // < 1V?
            // Mistura pobre!
            Rlambda = 0;
        }
        else {
            // Mistura rica!
            Rlambda = 1;
        }
    }

    // Calculo do erro
    erro_lambda = 0.5 - (float)Rlambda;

    if (((control_ant <= Min_Control) && (erro_lambda
        < 0)) // Houve saturacao do sinal de controle?
        || ((control_ant >= Max_Control) && (erro_lambda
        > 0)))
        flSaturacao = 1; // Seta a flag de saturacao!

    // Calcula o incremento da parte proporcional
    delta_control_P = KPlambda * (erro_lambda -
        erro_lambda_ant);

    // Calcula o incremento da parte integral
    Ts = 60.0 / (float)rotacao; // Período do
        controle = 1 volta do motor
    KIlambda = 12.0 / (float)rotacao; // Ganho Ki
    delta_control_I = KIlambda * Ts * erro_lambda;

    if (flSaturacao == 0) // Nao ocorreu saturacao? (
        Anti-Windup)
```

```
        control += (delta_control_P + delta_control_I
                    );

        // Limitacao da acao de controle (PWM)
        if (control > Max_Control)
            control = Max_Control;
        else if (control < Min_Control)
            control = Min_Control;
    }
}

else if (fabsf(delta_rotacao) >= MAX_DELTA_RPM_LAMBDA) {
    // Aceleracao / Desaceleracao detectada!
    flAceleracao = 1;
    LambdaTimer = 0; // Timer deve ser resetado!
    control = 0; // Zera a correcao acumulada!
}

}

else {
    control = 0; // Zera a correcao acumulada!
}

// Guarda os valores p/ o proximo laço de controle
erro_lambda_ant = erro_lambda;
control_ant = control;

return control;
}
```

A.1.10 Controle de aquecimento da sonda lambda

```

// Rotina que controla o aquecimento da sonda lambda e decrementa
// os timers associados
// OBS: Deve ser chamada a cada 10 ms (timer 1 do controle de RPM
// )
void heater_lambda() {
    static unsigned char flFirstTime = 1;
    static unsigned int HeaterTimer = 0;
    static unsigned char estadoAquecimento = 0; // 0-OFF; 1-ON

    // Decrementar o timer de atraso do controle lambda
    if (LambdaTimer > 0) {
        LambdaTimer--;
        if (LambdaTimer == 0) {
            if (!flLambdaAquecida)
                flLambdaAquecida = 1;
        }
    }

    // Decrementar timer do aquecimento da lambda
    if (HeaterTimer > 0)
        HeaterTimer--;

    if ((t_agua > 50.0) && (erro_hw == ERRO_HW_OK)) { // Motor
        quente?

        // Controle da tensao de aquecimento da sonda lambda
        // 5s ON + 0,5s OFF
        //
        //      -----      -----      -----      ----...
        //      |              | |              | |              | |
        //      |              | |              | |              | |
        //      ---|          |_|          |_|          |_|
        //      :              : :          : :          : :
        //      :              : :          : :          : :
        //      :<---5s--->: :<---5s--->: :<---5s--->: :
        //              <->          <->          <->
        //              0,5s          0,5s          0,5s
        //
        if (!estadoAquecimento) {
            // Aquecimento desligado!
            if (HeaterTimer == 0) { // Timer expirou?

```

```
        RELE_HEATER_LAMBDA = 1; // Liga aquecimento
        estadoAquecimento = 1; // Muda estado do
            aquecimento
        HeaterTimer = 500; // Aquecimento deve ficar
            ligado por 5s
    }
}
else {
    // Aquecimento ligado!
    if (HeaterTimer == 0) { // Timer expirou?
        RELE_HEATER_LAMBDA = 0; // Desliga aquecimento
        estadoAquecimento = 0; // Muda estado do
            aquecimento
        HeaterTimer = 50; // Aquecimento deve ficar
            desligado por 0,5s
    }
}

// Verificar se a tensao do sensor lambda caiu abaixo de
2.5V
if (lambda < 127) { // Tensao lambda < 2.5V?
    if (flFirstTime) { // Primeira vez que a tensao caiu
        abaixo de 2.5V?
        LambdaTimer = 2000; // Configura 20s para liberar
            o controle lambda
        flFirstTime = 0;
    }
}
}
else { // Motor frio ou erro de hardware! -> Resetar logica
    de aquecimento!
    flFirstTime = 1;
    LambdaTimer = 0;
    HeaterTimer = 0;
    estadoAquecimento = 0;
    flLambdaAquecida = 0;
    RELE_HEATER_LAMBDA = 0;
}
}
```

A.1.11 Controle de partida

```
// Rotina para controle da partida do motor
void controle_partida() {
    float Kref1;
    float Kref2;
    float ref_tps_partida; // Abertura Inicial p/ o motor pegar (
        pré partida)
    unsigned int delay_partida; // Atraso para segurar a ref da
        VB (estabilizacao da marcha lenta)

    // Bomba de Combustivel - Pressurizacao
    RELE_BOMBA_COMB = 1;
    delay_ms(1000);
    RELE_BOMBA_COMB = 0;

    // Ganho de injecao para partida com motor frio
    //
    //
    //      +20%  -----
    //                               : -
    //                               :  -
    //                               :   -
    //                               :    -----  0 %
    //      (sem ganho)
    //                               :      :
    //                               30°C   50°C
    //
    // Atraso do controle de partida para estabilizacao da marcha
    //      lenta
    //
    //
    //      5s  -----
    //                               : -
    //                               :  -
    //                               :   -
    //                               :    -----  1s
    //                               :      :
    //                               30°C   50°C
    //
    // Abertura Inicial da VB p/ o motor pegar na partida
    //
```

```

//
//   TPS_PARTIDA_FRIIO -----
//
//                               : -
//                               :  -
//                               :   -
//                               :    -----
//   TPS_PARTIDA
//                               :      :
//                               30°C   50°C
//
if (t_agua < 30.0) { // Motor frio?
    ganho_inj_partida = 0.2;
    delay_partida = 5000;
    ref_tps_partida = TPS_PARTIDA_FRIIO;
}
else if (t_agua < 50.0) { // Transição motor frio -> motor
    quente?
    ganho_inj_partida = (0.5 - 0.01*t_agua);
    delay_partida = (unsigned int)(11000.0 - 200.0*t_agua);
    ref_tps_partida = TPS_PARTIDA - (t_agua - 50.0)*(
        TPS_PARTIDA_FRIIO - TPS_PARTIDA)*0.05;
}
else { // Motor quente
    ganho_inj_partida = 0; // Sem ganho de injeção p/ partida
        a quente
    delay_partida = 1000;
    ref_tps_partida = TPS_PARTIDA;
}

// Constantes p/ mudar a escala da ref de torque p/ a escala
da VB
Kref1 = (TPS_MAX-ref_tps_partida) * 0.01;
Kref2 = ref_tps_partida;

timerPartida = DELAY_DESLIGAR_VB; // Habilita timer para
desligar a VB

while (rotacao < MIN_RPM_MLENTA) {
    // Libera a VB p/ seguir o pedal (ref. torque) enquanto a
partida não é acionada
    if (rotacao == 0) {
        ref_vb = Kref1*ref_torque + Kref2;
    }
}

```

```
        if (ref_torque < REF_TORQUE_MLENTA) {
            if (timerPartida == 0)
                flControlarVB = 0; // Desliga a VB
        }
        else {
            timerPartida = DELAY_DESLIGAR_VB; // Reseta o
            timer da partida
            flControlarVB = 1; // Ativa o controle da VB
        }
    }
    else {
        ref_vb = ref_tps_partida;
        timerPartida = DELAY_DESLIGAR_VB; // Reseta o timer
        da partida
        flControlarVB = 1; // Ativa o controle da VB
    }
    delay_ms(10);
}

delay_ms(delay_partida); // Segura a ref da VB para
    estabilizacao da marcha lenta
ganho_inj_partida = 0; // Remove o ganho de injecao da
    partida a frio
timerPartida = DELAY_MOTOR_REGIME; // Atraso para indicar
    motor em regime!
    flPosPartida = 1; // Partida do motor realizada com
        sucesso!
}
```

A.1.12 Cálculo da velocidade do veículo

```
// Calcula a velocidade do veiculo a cada 10ms
#define N_VELOCIDADE 20 // Numero de amostras p/ o calculo
void calcula_velocidade() {
    static float velocidade_ant[N_VELOCIDADE];
    unsigned char i; // Contador
    float soma_velocidade = 0;
    float periodo_vel_temp, velocidade_temp;
    static unsigned char flFirstTime = 1;

    // Zera o vetor na primeira utilizacao
    if (flFirstTime) {
        for (i = 0; i < N_VELOCIDADE; i++)
            velocidade_ant[i] = 0;

        flFirstTime = 0; // Reseta flag
    }

    // Calculo da velocidade
    periodo_vel_temp = (float)periodoSinalVelocidade;
    if (periodo_vel_temp > 0) {
        velocidade_temp = 1000000.0 / periodo_vel_temp; //
        // Velocidade calculada (km/h = Hz)
        if (velocidade_temp < 1.0)
            velocidade_temp = 0;
    }
    else
        velocidade_temp = 0;

    // Atualiza a posicao "0" com o valor atual
    velocidade_ant[0] = velocidade_temp;

    // Calcula a media das amostras
    for (i = 0; i < N_VELOCIDADE; i++)
        soma_velocidade += velocidade_ant[i];
    velocidade_medida = soma_velocidade / (float)N_VELOCIDADE;

    // Salva os valores p/ a proxima vez!
    for (i = N_VELOCIDADE - 1; i > 0; i--)
        velocidade_ant[i] = velocidade_ant[i-1];
}
```


A.1.13 Algoritmo para identificação de marcha

```
// Rotina para identificar a marcha atual, com base na rotacao e
// na velocidade do veiculo
// Retorna o valor da marcha engatada (1= Primeira marcha, 2 =
// Segunda marcha...)
// Retorna 0 se o veiculo estiver desengatado!
unsigned char identifica_marcha() {
    unsigned char i = 0; // contador
    float K_atual; // Relacao rotacao/velocidade atual
    unsigned char marcha_final = 0; // Assume-se inicialmente que
        o veiculo esta desengatado
    float erro_min = 65535.0; // Inicia com valor elevado!
    float erro_temp;

    // Constantes para a relação rotacao/velocidade de cada
    // marcha
    // OBS: Rotacao em RPM e velocidade em km/h
    static const float K[5] = { 115.3, 65.8, 46.8, 34.4, 27.0 };
    static const float tolerancia = 0.1; // faixa de tolerancia
        para comparacao (0.1 = 10%)

    if ((velocidade > 0) && (embreagem == 0)) { // Veiculo em
        movimento e embreagem solta?
        // Calcula a relacao rotacao/velocidade atual
        K_atual = (float)rotacao / velocidade_medida;

        // Varre todas as constantes...
        for (i = 0; i < 5; i++) {
            if ((K_atual < K[i]*(1.0 + tolerancia)) && (K_atual >
                K[i]*(1.0 - tolerancia))) {
                // Calcula o erro
                erro_temp = fabsf(K_atual - K[i]);
                if (erro_temp < erro_min) {
                    erro_min = erro_temp;
                    marcha_final = i+1;
                }
            }
        }
    }

    return marcha_final;
}
```

A.1.14 Algoritmo para *debounce* de sensores digitais

```
// Rotina p/ debouncing de sensores digitais
// OBS: Deve ser chamada a cada 10ms!
#define N_DEBOUNCE 4 // Numero de amostras utilizadas p/ debounce
void debounce_sensors() {
    // Armazena as amostras p/ debouncing
    // OBS: Cada bit representa o estado de um sensor:
    // 1 - Ativo; 0 - Inativo
    // 0bxxxxxxx
    // b0 = Acionamento da L15
    // b1 = Acionamento do cruzeiro
    // b2 = Freio
    // b3 = Embreagem
    // b4 = Falha no 33186 (Pino "SF")
    // b5 = Acionamento do botão de emergência
    static unsigned char db_sensors[N_DEBOUNCE];

    // Mascaras p/ manipulacao dos bits de cada sensor
    static const unsigned char ac_L15_mask = 0x01;
    static const unsigned char ac_cruzeiro_mask = 0x02;
    static const unsigned char freio_mask = 0x04;
    static const unsigned char embreagem_mask = 0x08;
    static const unsigned char sf_33186_mask = 0x10;
    static const unsigned char ac_emergencia_mask = 0x20;

    static const unsigned char ac_L15_shift = 0;
    static const unsigned char ac_cruzeiro_shift = 1;
    static const unsigned char freio_shift = 2;
    static const unsigned char embreagem_shift = 3;
    static const unsigned char sf_33186_shift = 4;
    static const unsigned char ac_emergencia_shift = 5;

    // Indica os sensores que mudaram de estado
    unsigned char sensors_changed = 0;

    unsigned char i, j; // contador
    static unsigned char flFirstTime = 1;

    // Armazena o estado anterior do respectivo sinal
    static unsigned char freio_ant = 0;
    static unsigned char embreagem_ant = 0;
```

```
static unsigned char ac_cruzeiro_ant = 0;
static unsigned char estado_L15_ant = 0;
static unsigned char estado_SF_33186_ant = 0;
static unsigned char estado_emergencia_ant = 0;

// Zera o vetor na primeira utilizacao
if (flFirstTime) {
    for (i = 0; i < N_DEBOUNCE; i++)
        db_sensors[i] = 0;

    flFirstTime = 0; // Reseta flag
}

// Atualiza cada bit da posicao "0" com a leitura de cada
sensor
db_sensors[0] = 0;
if (SINAL_L15) // Sinal ativo "alto"
    db_sensors[0] |= ac_L15_mask;
if (!AC_CRUZEIRO) // Sinal ativo "baixo"
    db_sensors[0] |= ac_cruzeiro_mask;
if (SINAL_FREIO) // Sinal ativo "alto"
    db_sensors[0] |= freio_mask;
if (!SINAL_EMBREAGEM) // Sinal ativo "baixo"
    db_sensors[0] |= embreagem_mask;
if (!SINAL_SF_33186) // Sinal ativo "baixo"
    db_sensors[0] |= sf_33186_mask;
if (!AC_BOTAO_EMERGENCIA) // Sinal ativo "baixo"
    db_sensors[0] |= ac_emergencia_mask;

// Debouncing das medidas armazenadas
for (i = 0; i < N_DEBOUNCE; i++) {
    if (i < N_DEBOUNCE-1) {
        for (j = i+1; j < N_DEBOUNCE; j++)
            sensors_changed |= (db_sensors[i] ^ db_sensors[j]);
    }
}

// Verifica os sensores que possuem todas as medidas "
estaveis"
if ((sensors_changed & ac_L15_mask) == 0)
```

```
    estado_L15 = ((db_sensors[0] & ac_L15_mask) >>
        ac_L15_shift);
if ((sensors_changed & ac_cruzeiro_mask) == 0)
    ac_cruzeiro = ((db_sensors[0] & ac_cruzeiro_mask) >>
        ac_cruzeiro_shift);
if ((sensors_changed & freio_mask) == 0)
    freio = ((db_sensors[0] & freio_mask) >> freio_shift);
if ((sensors_changed & embreagem_mask) == 0)
    embreagem = ((db_sensors[0] & embreagem_mask) >>
        embreagem_shift);
if ((sensors_changed & sf_33186_mask) == 0)
    estado_SF_33186 = ((db_sensors[0] & sf_33186_mask) >>
        sf_33186_shift);
if ((sensors_changed & ac_emergencia_mask) == 0)
    estado_botao_emergencia = ((db_sensors[0] &
        ac_emergencia_mask) >> ac_emergencia_shift);

// Acoes a serem tomadas quando houver mudanca de estado dos
// sensores
if (estado_L15 != estado_L15_ant) {
    if (estado_L15 == 0) { // Linha 15 acabou de ser
        desligada?
        // Desliga atuadores do sistema
        prepareShutdown();

        // Reseta o MCU via Watchdog!
        execReset();
    }
}

if (ac_cruzeiro != ac_cruzeiro_ant) {
    if (ac_cruzeiro != 0) { // Cruzeiro acabou de ser
        acionado?
        // Cruzeiro acionado!
        modo_cruzeiro = 1;
    }
    else {
        // Cruzeiro desacionado!
        modo_cruzeiro = 0;
    }
}

if (estado_SF_33186 != estado_SF_33186_ant) {
    if (estado_SF_33186 != 0) { // Pino SF "ativo"?
```

```
        if (fl33186Init) { // Verifica somente se o 33186 ja
            foi inicializado!
            rotina_erro_hw(ERRO_HW_33186); // 33186 com falha
            !!
            flControlarVB = 0; // Desliga o controle da VB (
                ir para posicao de Limp Home)!
        }
    }
}

if (estado_botao_emergencia != estado_emergencia_ant) {
    if ((estado_botao_emergencia != 0) && (estado_L15 != 0))
    { // Botao de emergencia "ativo" e linha 15 ligada?
        // Desliga todos os atuadores!!!
        prepareShutdown();

        // Mantem o LED de erro de HW aceso p/ notificar o
            usuario!
        LEDGER3 = 0;

        // Espera o reset da L15
        espera_L15();

        // Reseta o MCU via Watchdog!
        execReset();
    }
}

// Desloca as amostras p/ a proxima chamada
for (i = N_DEBOUNCE-1; i > 0; i--)
    db_sensors[i] = db_sensors[i-1];

// Salva o estado dos sensores
estado_L15_ant = estado_L15;
ac_cruzeiro_ant = ac_cruzeiro;
freio_ant = freio;
embreagem_ant = embreagem;
estado_SF_33186_ant = estado_SF_33186;
estado_emergencia_ant = estado_botao_emergencia;
}
```

A.1.15 Filtro da referência de posição da borboleta eletrônica

```
// Filtro para impedir variacoes bruscas na VB quando o veiculo
// estiver engatado!
float filtroRefVB(float ref_vb_atual) {
    static float ref_vb_anterior = TPS_MLENTA_FRIIO;
    float ref_vb_filtrado;
    float delta_abertura = 65535.0; // Variacao maxima da
    // abertura da VB
    float delta_fechamento = 65535.0; // Variacao maxima do
    // fechamento da VB
    static const float periodo = 10.0; // Período (em ms) do
    // filtro, ou seja, a cada 10 ms a rotina é chamada.

    ref_vb_filtrado = ref_vb_atual;

    if (t_agua < 50.0) { // Motor frio?
        // Abertura da VB deve ser bem lenta p/ nao afogar o
        // motor!
        delta_abertura = (TPS_MAX - TPS_MLENTA) * periodo /
            5000.0; // 5000ms de duracao p/ a abertura de
            TPS_MLENTA a TPS_MAX

        if (ref_vb_anterior > TPS_MAX_TRANCO) // Libera
            // fechamento "rapido" ate a posicao "TPS_MAX_TRANCO"
            delta_fechamento = ref_vb_anterior - TPS_MAX_TRANCO;
        else // Abaixo da posicao "TPS_MAX_TRANCO" o fechamento
            // passa a ser mais lento!
            delta_fechamento = (TPS_MAX_TRANCO - TPS_MLENTA) *
                periodo / 500.0; // 500ms de duracao p/ o
                // fechamento de TPS_MAX_TRANCO a TPS_MLENTA
    }

    else if (marcha_engatada != 0) { // Veiculo engatado?
        if (ref_vb_anterior < TPS_MAX_TRANCO) // Inicio da
            // abertura da VB deve ser lento para nao dar tranco!
            delta_abertura = (TPS_MAX_TRANCO - TPS_MLENTA) *
                periodo / 500.0; // 500ms de duracao p/ a abertura
                // de TPS_MLENTA a TPS_MAX_TRANCO
        else // Libera uma abertura um pouco mais rapida!
            delta_abertura = (TPS_MAX - TPS_MAX_TRANCO) * periodo
                / 800.0; // 800ms de duracao p/ a abertura de
                TPS_MAX_TRANCO a TPS_MAX
    }
}
```

```
    if (ref_vb_anterior > TPS_LIMP_HOME) // Libera fechamento
        "rapido" ate a posicao de LH
        delta_fechamento = ref_vb_anterior - TPS_LIMP_HOME;
    else // Abaixo do LH o fechamento passa a ser mais lento!
        delta_fechamento = (TPS_LIMP_HOME - TPS_MLENTA) *
            periodo / 500.0; // 500ms de duracao p/ o
            fechamento de TPS_LIMP_HOME a TPS_MLENTA
}

else if ((velocidade > 0) && (embreagem != 0)) { // Troca de
marcha?
    if (ref_vb_anterior > TPS_MLENTA_FRIIO) // Libera
        fechamento "rapido" ate a posicao "TPS_MLENTA_FRIIO"
        delta_fechamento = ref_vb_anterior - TPS_MLENTA_FRIIO;
    else // Abaixo da posicao "TPS_MLENTA_FRIIO" o fechamento
        passa a ser mais lento!
        delta_fechamento = (TPS_MLENTA_FRIIO - TPS_MLENTA) *
            periodo / 2000.0; // 2s de duracao p/ o fechamento
            de TPS_MLENTA_FRIIO a TPS_MLENTA
}

if (modo_operacao != MODO_OPERACAO_ESPORTIVO) { // Nao eh
modo de operacao esportivo?
    if ((ref_vb_atual - ref_vb_anterior) > delta_abertura)
        ref_vb_filtrado = ref_vb_anterior + delta_abertura;
        // Suaviza abertura
    else if ((ref_vb_anterior - ref_vb_atual) >
        delta_fechamento)
        ref_vb_filtrado = ref_vb_anterior - delta_fechamento;
        // Suaviza fechamento
}

// Guarda o valor de ref VB para a proxima chamada da rotina
ref_vb_anterior = ref_vb_filtrado;

return ref_vb_filtrado;
}
```

A.2 Bloco de Sincronismo

A.2.1 Atualização dos parâmetros de injeção e ignição

```
// Rotina para atualizar os parametros de injecao/ignicao com os
// valores recebidos do GER
void atualizaParametros() {
    unsigned int CRC_calculado = 0xFFFF;
    unsigned int CRC_enviado = valor[SPI_MAX-1];

    if (spi_ctr == SPI_MAX) { // Ja recebeu todos os parametros
        do GER?

        // Calcula o CRC16 (CCITT) dos parametros fornecidos (
        // excluindo a ultima posicao, que contem o CRC enviado)
        CRC_calculado = calcula_crc16((unsigned char *)&valor[0],
            ((SPI_MAX-1) << 1));

        if (CRC_calculado == CRC_enviado) { // CRC bateu?
            // Atualiza adiantamentos e tempos
            ig_dente_14 = (unsigned char)(valor[0] & 0xFF);
            if ((ig_dente_14 < 1) || (ig_dente_14 > 58)) { //
                Verificacao de seguranca...
                ig_dente_14 = 255;
                ig_dente_23 = 255;
            }
            else {
                ig_dente_23 = ig_dente_14 + 30;
                if (ig_dente_23 > 60) ig_dente_23 -= 60;
            }

            ij_dente_14 = (unsigned char)((valor[0] >> 8) & 0xFF)
                ; // 0 dente de inj foi enviado no byte mais sig.
                do SPI (SPIxDRH)
            if ((ij_dente_14 < 1) || (ij_dente_14 > 58)) { //
                Verificacao de seguranca...
                ij_dente_14 = 255;
                ij_dente_23 = 255;
            }
            else {
                ij_dente_23 = ij_dente_14 + 30;
                if (ij_dente_23 > 60) ij_dente_23 -= 60;
            }
        }
    }
}
```



```
    }  
    ig_tmr_disparo = valor[1];  
    ij_tmr_disparo = valor[2];  
    tempo_inj = valor[3];  
    tempo_inj_extra = valor[4];  
    tempo_bob = valor[5];  
    rotacao = valor[6];  
    cutoff = (unsigned char)(valor[7] & 0xFF);  
  
    erro = erro & ~0b10000000;  
} // if (CRC_calculado == CRC_enviado)  
else {  
    if (crc_mismatches < 255)  
        crc_mismatches++;  
}  
  
    erro = erro & ~0b01000000;  
} // if (spi_ctr == SPI_MAX)  
}
```

A.2.2 Interrupção do *timer* que controla os sinais de injeção 1 e 4

```
__interrupt void TIMER_INJ14_PIT1(void)
{
    PITTF = PITTF_PTF1_MASK; // Limpa flag PTF1
    PITINTE_PINTE1 = 0; // Desabilita int timer 1
    EnableInterrupts; // Limpa bit "I" -> Interrupcoes de
        prioridade maior podem interromper a atual!

    /* Pulsos de injecao p sequencial*/
    switch(ij14_pos) {

        case 0:
            break;

        case 10:
            SINAL_IJ1 = 0; /* Baixa o Sinal da Injecao 1 */
            LEDSINC3 = ~SINAL_IJ1; // 0 estado do led "SINC
                3" (ativo baixo) acompanha o comando do Injetor
                1

            volta14 = ~volta14;

            // Verifica a logica da fase...
            // Neste caso o sinal de injecao 1 SEMPRE desce
            DEPOIS da falha
            if (sinal_fase == 0) {
                // A fase estava errada!!!
                volta14 = ~volta14;
                if (correcoes_fase < 255)
                    correcoes_fase++;
            }

            ij14_pos = 0;
            break;

        case 40:
            SINAL_IJ4 = 0; /* Baixa o Sinal da Injecao 4 */

            volta14 = ~volta14;

            ij14_pos = 0;
```

```
        break;

    case 12:
        if (tempo_inj_aux > 50) {
            SINAI_IJ4 = 0; // Desliga a injeção auxiliar
            if (!cutoff) {
                SINAI_IJ1 = 1; // Garante que o sinal de
                    injeção 1 ainda está ativo
                LEDSINC3 = ~SINAI_IJ1; // O estado do led
                    "SINC 3" (ativo baixo) acompanha o
                    comando do Injetor 1
            }

            // Mantem o sinal de injeção 1 ativo para
                completar o tempo de inj
            PITLD1 = tempo_inj_aux;
            PITFLT_PFLT1 = 1;
            PITTF = PITTF_PTF1_MASK; // Limpa flag PTF1
            PITINTE_PINTE1 = 1; // Habilita interrupção
                timer 1

            ij14_pos = 10; // Próxima interrupção vai
                desligar o sinal
        }
        else {
            SINAI_IJ1 = 0;
            LEDSINC3 = ~SINAI_IJ1; // O estado do led "
                SINC 3" (ativo baixo) acompanha o comando
                do Injetor 1
            SINAI_IJ4 = 0;
            volta14 = ~volta14;
            ij14_pos = 0;
        }
        break;

    case 42:
        if (tempo_inj_aux > 50) {
            SINAI_IJ1 = 0; // Desliga a injeção auxiliar
            LEDSINC3 = ~SINAI_IJ1; // O estado do led "
                SINC 3" (ativo baixo) acompanha o comando
                do Injetor 1
            if (!cutoff)
```

```
        SINHAL_IJ4 = 1; // Garante que o sinal de
                        injeção 4 ainda está ativo

        // Mantem o sinal de injeção 1 ativo para
        // completar o tempo de inj
        PITLD1 = tempo_inj_aux;
        PITFLT_PFLT1 = 1;
        PITTF = PITTF_PTF1_MASK; // Limpa flag PTF1
        PITINTE_PINTE1 = 1; // Habilita interrupção
        timer 1

        ij14_pos = 40; // Próxima interrupção vai
        desligar o sinal
    }
    else {
        SINHAL_IJ1 = 0;
        LEDSINC3 = ~SINAL_IJ1; // 0 estado do led "
        SINC 3" (ativo baixo) acompanha o comando
        do Injetor 1
        SINHAL_IJ4 = 0;
        volta14 = ~volta14;
        ij14_pos = 0;
    }
    break;

case 1:
    if (tempo_inj > 50) {
        if (!cutoff) {
            SINHAL_IJ1 = 1; /* Ativa Sinal da injeção
            1 */
            LEDSINC3 = ~SINAL_IJ1; // 0 estado do led
            "SINC 3" (ativo baixo) acompanha o
            comando do Injetor 1
        }

        if (tempo_inj_extra > 50) {
            if (!cutoff)
                SINHAL_IJ4 = 1; // Banco a banco

            // Utiliza uma pequeno tempo adicional
            para banco a banco
            PITLD1 = tempo_inj_extra;
```

```
PITFLT_PFLT1 = 1;
PITTF = PITTF_PTF1_MASK; // Limpa flag
                        PTF1
PITINTE_PINTE1 = 1; // Habilita
                        interrupcao timer 1

ij14_pos = 12; // Proxima interrupcao vai
                        desligar o sinal auxiliar

}
else { // Somente sequencial
    /* Conta e entao baixa o sinal de injecao
       por interrupcao */
    PITLD1 = tempo_inj;
    PITFLT_PFLT1 = 1;
    PITTF = PITTF_PTF1_MASK; // Limpa flag
                        PTF1
    PITINTE_PINTE1 = 1; // Habilita
                        interrupcao timer 1

    ij14_pos = 10; // Proxima interrupcao vai
                        desligar o sinal

}
}
else {
    volta14 = ~volta14;
    ij14_pos = 0;
}
break;

case 4:
    if (tempo_inj > 50) {
        if (!cutoff)
            SINAI_IJ4 = 1; /* Ativa Sinal da injecao
                            4 */

        if (tempo_inj_extra > 50) {
            if (!cutoff) {
                SINAI_IJ1 = 1; // Banco a banco
                LEDSIINC3 = ~SINAI_IJ1; // O estado do
                        led "SINC 3" (ativo baixo)
                        acompanha o comando do Injetor 1
            }
        }
    }
}
```

```
    }

    // Utiliza uma pequeno tempo adicional
    // para banco a banco
    PITLD1 = tempo_inj_extra;
    PITFLT_PFLT1 = 1;
    PITTF = PITTF_PTF1_MASK; // Limpa flag
    PTF1
    PITINTE_PINTE1 = 1; // Habilita
    interrupcao timer 1

    ij14_pos = 42; // Proxima interrupcao vai
    desligar o sinal auxiliar

}

else { // Somente sequencial
    /* Conta e entao baixa o sinal de injecao
    por interrupcao */
    PITLD1 = tempo_inj;
    PITFLT_PFLT1 = 1;
    PITTF = PITTF_PTF1_MASK; // Limpa flag
    PTF1
    PITINTE_PINTE1 = 1; // Habilita
    interrupcao timer 1

    ij14_pos = 40; // Proxima interrupcao vai
    desligar o sinal

}

}

else {
    volta14 = ~volta14;
    ij14_pos = 0;
}

break;

}

erro = erro & ~0b00010000;
}
```

A.3 Bloco de Comunicação

A.3.1 Interrupção da recepção de dados da UART (processamento no XGATE)

```
__interrupt void SCIO_ISR_XGATE(void)
{
    static unsigned char char_serial = 0;
    static unsigned char char_serial_ant = 0;

    // Interrupcao da Serial - Comunicacao com o PC
    SCIOCR2_RIE = 0; // Desabilita interrupcao de RX da serial
    SCIOSR1 = SCIOSR1; // Limpar a flag de recepcao (RDRF = 0)
    char_serial = SCIODRL; // pega o dado da serial

    if (char_serial_ant == 'R') { // Comando de RETORNO de variável
        vel

        switch(char_serial) {

            case '1': // "R1" - Leitura das versoes de firmware
                SCIO_Write(fw_ver_ger);
                SCIO_Write(fw_ver_com);
                SCIO_Write(fw_ver_sinc);
                break;

            case '2': // "R2" - Manda todos os parametros
                SCIO_Write_16bits(rotacao);
                SCIO_Write_16bits(pedal);
                SCIO_Write_16bits(valvula);
                SCIO_Write_16bits(ref_torque);
                SCIO_Write_16bits(map);
                SCIO_Write(tar);
                SCIO_Write(tagua);
                SCIO_Write(lambda);
                SCIO_Write(bateria);
                SCIO_Write_16bits(ref_vb);
                SCIO_Write(ij_dente_14);
                SCIO_Write_16bits(ij_disparo);
                SCIO_Write(ig_dente_14);
                SCIO_Write_16bits(ig_disparo);
                SCIO_Write_16bits(tempo_inj);
                SCIO_Write_16bits(tempo_bob);
```

```
        SCIO_Write(correcoes_fase);
        SCIO_Write(erro_ger_sw);
        SCIO_Write(erro_ger_hw);
        SCIO_Write(erro_sinc_sw);
        SCIO_Write(velocidade);
        SCIO_Write(marcha_engatada);
        SCIO_Write(freio);
        SCIO_Write(ref_velocidade);
        SCIO_Write(modos_cruzeiro);
        SCIO_Write(embreagem);
        SCIO_Write_16bits(fator_inj_10000);
        SCIO_Write_16bits(pwm_vb_10);
        SCIO_Write(modos_ArCond);
        SCIO_Write(nivel_comb);
        SCIO_Write_16bits(delta_rotacao);
        SCIO_Write_16bits(rotacao_mlenta);
        SCIO_Write(luzes_painel);
        SCIO_Write(portas_veiculo);
        SCIO_Write((unsigned char)((hodometro >> 16) & 0xFF));
        SCIO_Write((unsigned char)((hodometro >> 8) & 0xFF));
        SCIO_Write((unsigned char)(hodometro & 0xFF));
        SCIO_Write(crc_mismatches);
        SCIO_Write_16bits(velocidade_medida);
        SCIO_Write(t_ambiente);
        break;

    case '3': // "R3" - Manda apenas os parametros do
        ensaio de identificacao do cruzeiro
        SCIO_Write_16bits(rotacao);
        SCIO_Write_16bits(ref_torque);
        SCIO_Write(velocidade);
        SCIO_Write(marcha_engatada);
        SCIO_Write(freio);
        SCIO_Write(embreagem);
        SCIO_Write_16bits(velocidade_medida);
        break;

    case '4': // "R4" - Manda apenas os parametros do
        ensaio de identificacao da VB
        SCIO_Write_16bits(valvula);
```



```
        SCIO_Write_16bits(pwm_vb_10);
        break;

    case '5': // "R5" - Manda apenas os parametros do
        ensaio de identificacao da Marcha Lenta
        SCIO_Write_16bits(rotacao);
        SCIO_Write_16bits(valvula);
        SCIO_Write_16bits(ref_vb);
        break;
    }
}

else if (char_serial_ant == 'E') { // Comando de ENVIO de
    variável
    switch(char_serial) {

        case '1': // "E1" - Modo pedal "simulado" (True/False
            )
            modo_pedal_simulado = SCIO_Read();
            break;

        case '2': // "E2" - Modo de operação
            modo_operacao = SCIO_Read();
            break;

        case '3': // "E3" - Leitura do pedal "simulado"
            pedal_simulado = SCIO_Read();
            break;

        case '4': // "E4" - Leitura da ref. de rotação da
            marcha lenta (motor quente)
            modo_ref_mlenta = SCIO_Read();
            break;
    }
}

char_serial_ant = char_serial;
SCIOCR2_RIE = 1; // Habilita interrupcao de RX da serial
}
```

A.3.2 Interrupção da recepção de dados da CAN

```
__interrupt void CAN0_RX(void)
{
    unsigned int msg_id = 0;
    unsigned char msg_length, index;
    unsigned char msg_data[8];
    unsigned char modo_ArCondTemp;

    CANORIER_RXFIE = 0; // Desabilita Int. CAN RX
    EnableInterrupts; // Limpa bit "I" -> Interrupcoes de
        prioridade maior podem interromper a atual!

#ifdef PINOS_TESTE
    IOCOM2 = 1; // Para testes...
#endif

    if (CANORXIDR1_IDE != 0) { // Formato estendido??

        #ifdef PINOS_TESTE
            IOCOM2 = 0; // Para testes...
        #endif

        CANORFLG = CANORFLG_RXF_MASK; // Limpa flag de RX
        CANORIER_RXFIE = 1; // Habilita Int. CAN RX

        return; // Nunca deveria entrar aqui!! Se entrar é porque
            tem problema no filtro de RX!
    }

    // Leitura do ID da mensagem recebida (11 bits: ID10 a ID0)
    msg_id = (unsigned int)(CANORXIDR0 & 0xFF); // ID10 a ID3
    msg_id = msg_id << 3;
    msg_id |= (unsigned int)((CANORXIDR1 >> 5) & 0xFF); // ID2 a
        ID0

    // Leitura do tamanho da mensagem recebida
    msg_length = CANORXDLR;

    // Nunca deveria receber mais do que 8 bytes, mas por
        seguranca...
    if (msg_length > 8)
```

```
msg_length = 8;

// Extracao da mensagem recebida
for (index = 0; index < msg_length; index++)
    msg_data[index] = *(&CANORXDSR0 + index);

// Limpa flag de RX
CANORFLG = CANORFLG_RXF_MASK;

// Verifica se o ID da mensagem é de interesse
switch (msg_id) {

    #ifdef PEDAL_VIA_CAN // Controle do pedal via CAN foi
        habilitado?
    case 0x200: // Comandos enviados pela ECU do Adaptive
        Cruise Control (ACC)
        if (msg_length >= 4) { // Foram recebidos pelo menos
            4 bytes?
            // Confere o token antes de atualizar qualquer
                parametro!
            // OBS: Token = "ECU"
            if ((msg_data[0] == 'E') && (msg_data[1] == 'C')
                && (msg_data[2] == 'U')) {
                // Token valido!
                // Atualiza o valor do pedal "simulado"
                pedal_simulado = msg_data[3];
            }
        }
        break;

    case 0x201: // Comandos enviados pela ECU do Adaptive
        Cruise Control (ACC)
        if (msg_length >= 6) { // Foram recebidos pelo menos
            6 bytes?
            // Confere o token antes de atualizar qualquer
                parametro!
            // OBS: Token = "ECU"
            if ((msg_data[0] == 'E') && (msg_data[1] == 'C')
                && (msg_data[2] == 'U')) {
                // Token valido!
                // Atualiza as configuracoes da ECU
```

```
        modo_pedal_simulado = msg_data[3]; // Modo
            pedal simulado
        modo_operacao = msg_data[4]; // Modo de
            operacao
        modo_ref_mlenta = msg_data[5]; // Modo Ref.
            Marcha Lenta
    }
}
break;
#endif

case 0x320:
    if (msg_length >= 7) { // Foram recebidos pelo menos
        7 bytes?
        // Leitura do nivel de combustivel do barramento
            CAN
        nivel_comb = msg_data[2]; // Unidade: Litros

        // Leitura do acionamento do freio de mao
        if (msg_data[1] & 0b00000010)
            freio_estac = 1; // Freio de estacionamento
                acionado
        else
            freio_estac = 0; // Freio de estacionamento
                desacionado

        // Leitura da luz de bateria
        if (msg_data[1] & 0b00000100)
            luz_bateria = 1; // Luz da bateria ligada
        else
            luz_bateria = 0; // Luz da bateria desligada

        // Leitura da velocidade do barramento CAN
        // OBS: Velocidade está multiplicada por 0.75 na
            mensagem!
        velocidade = (unsigned char)((float)msg_data[6] *
            1.333); // Unidade: km/h
    }
    break;

case 0x420:
```

```
    if (msg_length >= 3) { // Foram recebidos pelo menos
        3 bytes?
        // Leitura da temperatura ambiente
        t_ambiente = (signed char)(((float)msg_data[2] *
            0.5) - 50.0); // Unidade: °C
    }
    break;

case 0x470:
    if (msg_length >= 3) { // Foram recebidos pelo menos
        3 bytes?
        // Leitura da luzes do painel (Setas, pisca
            alerta e luz de ré)
        luzes_painel = (msg_data[0] & 0b00101011); //
            Seta esquerda ligada

        // Leitura da luz de backlight
        if (msg_data[2] != 0)
            luzes_painel |= 0b10000000;

        // Adiciona informacao das luzes restantes
        if (freio_estac)
            luzes_painel |= 0b00010000; // Luz do freio
            de estacionamento
        if (luz_bateria)
            luzes_painel |= 0b01000000; // Luz da bateria

        // Leitura do estado das portas do veiculo
        portas_veiculo = (msg_data[1] & 0b00111111);
    }
    break;

case 0x520:
    if (msg_length >= 8) { // Foram recebidos pelo menos
        8 bytes?
        // Leitura do hodometro (unidade: km)
        hodometro = (((unsigned long)msg_data[7] & 0xFF)
            << 16) // MSB
            | (((unsigned long)msg_data[6] & 0xFF) << 8)
            | (((unsigned long)msg_data[5] & 0xFF); // LSB
    }
    break;
```

```
case 0x5E0:
    if (msg_length >= 7) { // Foram recebidos pelo menos
        7 bytes?
        // Leitura do acionamento do Ar Condicionado (
        Painel do Motorista)
        // OBS: Informação no bit 1 -> 1 ligado; 0
        desligado
        modo_ArCondTemp = (msg_data[6] & 0b00000010) >>
            1;
        if ((modo_ArCond == 0) && (modo_ArCondTemp != 0))
            // Ar Condicionado acabou de ser acionado?
            timerArCond = 200; // Atraso de 200*10ms = 2s
            para ligar o compressor!
        modo_ArCond = modo_ArCondTemp;

    }
    break;

case 0x7DF:
case 0x7E0:
    for (index = 0; index < 8; index++)
        OBDdata[index] = msg_data[index]; // Pega o dado
        recebido
    OBDflag = 1;
    break;

default:
    // Nunca deveria entrar aqui!! Se entrar é porque tem
    problema no filtro de RX!
    break;
}

#ifdef PINOS_TESTE
IOCOM2 = 0; // Para testes...
#endif

CANORIER_RXFIE = 1; // Habilita Int. CAN RX
}
```

APÊNDICE B – *Software*

A seguir foram disponibilizados trechos do código fonte da aplicação ECU v2.0 Monitor, que foi desenvolvida em linguagem C# para monitoramento da ECU v2.0 e para automatizar o ensaio de identificação do sistema que foi modelado para uso no projeto do controle de cruzeiro.

B.1 Rotina para ativação e desativação do monitoramento da ECU

```
// Controla a aquisicao de valores do COM
// estado = true -> ativa aquisicao de valores
// estado = false -> desativa aquisicao de valores
private void Aquisicao(bool estado)
{
    if (estado == true)
    {
        // Ativar aquisicao de dados
        if (medindo == false) // confirma que a aquisicao estava
            parada
        {
            // Insere formato inicial da string de log
            logger.PrepareFileHeader();

            // Zera os registros de tempo
            t_Timer = t_Aquisicao = t_GUI = t_Logger = 0;
            t0_gauge = tf_gauge = 0;

            // Atualizar periodo de geracao de dados
            periodo_aquisicao = Convert.ToInt32(periodo_comboBox.
                SelectedItem.ToString().Replace(" ms", ""));
            timerAquisicao.Period = periodo_aquisicao;
            computadorBordo.SetPeriodoAquisicao(periodo_aquisicao
                );

            // Libera timer para atualizar os parâmetros
            timerAquisicao.Start();

            // Libera a atualização do painel
            panelManager.Start();
        }
    }
}
```

```
        medindo = true;
        lervalores_button.Text = "PARAR"; // Aproveita o
            mesmo botao para parar a aquisicao

        // Libera counter de debug
        stopwatch.Restart();

        // Liga a barra de progresso
        progressBar1.MarqueeAnimationSpeed = 10;
    }
}
else
{
    // Parar aquisicao de dados
    if (medindo == true) // confirma que a aquisicao estava
        em andamento
    {
        // Desabilita timer para parar atualizacao dos
            parametros
        timerAquisicao.Stop();

        // Desabilita a atualizaçao do painel
        panelManager.Stop();

        medindo = false;
        lervalores_button.Text = "LER DADOS"; // Reseta texto
            do botao de aquisicao

        // Parar counter de debug
        stopwatch.Stop();

        // Desliga a barra de progresso
        progressBar1.MarqueeAnimationSpeed = 0;
    }
}
}
```


B.2 Interrupção do *timer* de controle do monitoramento da ECU

```
// Rotina chamada quando o timer de aquisicao estourar
private void timerAquisicaoEvent(object sender, EventArgs e)
{
    bool result;

    // Eleva temporariamente a prioridade do processo!
    Process.GetCurrentProcess().PriorityBoostEnabled = true;

    timerMutex.WaitOne(); // Acesso ao mutex do timer de
        amostragem

    // Executa o código abaixo na Thread principal da aplicação
    Invoke((MethodInvoker) delegate
    {
        // Registra o tempo decorrido
        t_Timer = stopwatch.ElapsedMilliseconds; // em ms

        // Requisita valores dos Sensores/Variaveis da ECU v2.0
        result = parametros.RequisitarCOM(comando_aquisicao);
        if (result == false) // Erro na comunicacao?
        {
            Aquisicao(false);
            MessageBox.Show("Não há dados na Serial!", "Erro de
                Comunicacao!", MessageBoxButtons.OK, MessageBoxIcon
                .Warning);
        }

        // Registra o tempo decorrido
        t_Aquisicao = stopwatch.ElapsedMilliseconds; // em ms

        // Atualiza os parâmetros de computador de bordo
        computadorBordo.AtualizarParametros();

        // Escreve na tela todos os parametros recebidos do COM
        GUILogger();

        // Escreve em arquivo parte dos parametros recebidos do
        COM
        FileLogger();
    });
}
```

```
        // Pisca "LED" de status
        statusLED.Checked = !statusLED.Checked;
    });

    timerMutex.ReleaseMutex(); // Libera o mutex
}
```

B.3 Interrupção do *timer* de controle do ensaio de identificação

```
private static int starting_counter = 0; // conta os 5s para
    liberar o ensaio
// Rotina chamada quando o timer de 10ms estourar
private void Timer10msEvent(object sender, EventArgs e)
{
    double pedal;

    timer10msMutex.WaitOne(); // Espera acesso ao mutex

    // Executa o código abaixo na Thread principal da aplicação
    Invoke((MethodInvoker)delegate
    {
        switch (estado_ensaio)
        {
            case EstadoEnsaio.STATE_IDLE:
                // Desabilita o fechamento da janela enquanto o
                // ensaio estiver em andamento!
                this.ControlBox = false;

                // Assumir controle do pedal do veículo!
                if (pParametros.EnviaCOM(PARAMETROS_ECU.
                    ID_MODO_PEDAL_SIMULADO, 1) == false)
                {
                    // Falha de comunicação! -> Abortar ensaio de
                    // identificacao!
                    estado_ensaio = EstadoEnsaio.STATE_COMFAILURE
                        ;
                    break;
                }

                // Aplicar ref. de torque minima (= PEDAL_MIN)
                if (pParametros.EnviaCOM(PARAMETROS_ECU.
                    ID_PEDAL_SIMULADO, pedal_min) == false)
                {
                    // Falha de comunicação! -> Abortar ensaio de
                    // identificacao!
                    estado_ensaio = EstadoEnsaio.STATE_COMFAILURE
                        ;
                    break;
                }
            }
        }
    });
}
```

```
refTorque_atual = 0;

// Inicia contagem dos 5s para começar o teste
starting_counter = 5;
stopwatch.Restart();
estado_ensaio = EstadoEnsaio.STATE_STARTING;
status_textBox.Text = ST_STARTING +
    starting_counter.ToString();
break;

case EstadoEnsaio.STATE_STARTING:
    // Atualiza contagem inicial para inicio (5s)
    if (stopwatch.ElapsedMilliseconds >= 1000)
    {
        starting_counter--;
        status_textBox.Text = ST_STARTING +
            starting_counter.ToString();
        if (starting_counter == 0)
        {
            // Ja contou 5s -> mudar para o proximo
            estado!
            estado_ensaio = EstadoEnsaio.
                STATE_WAITING_STEP;
            status_textBox.Text = ST_WAITING_STEP;
        }
        stopwatch.Restart();
    }
    break;

case EstadoEnsaio.STATE_WAITING_STEP:
    if (stopwatch.ElapsedMilliseconds >=
        instante_degrau)
    {
        // Aplicar o degrau no pedal do acelerador!!
        // OBS: É necessário converter a escala da
        // ref. de torque [0 a 100%] p/ a escala do
        // pedal simulado [pedal_min a pedal_max].
        pedal = (double)pedal_min + (double)(
            pedal_max - pedal_min) * amplitude_degrau *
            0.01;
        if (pParametros.EnviaCOM(PARAMETROS_ECU.
            ID_PEDAL_SIMULADO, (int)pedal) == false)
```

```
{  
    // Falha de comunicação! -> Abortar  
    ensaio de identificacao!  
    estado_ensaio = EstadoEnsaio.  
        STATE_COMFAILURE;  
    break;  
}  
refTorque_atual = amplitude_degrau;  
  
// Mover a maquina para o proximo estado!  
estado_ensaio = EstadoEnsaio.  
    STATE_STEP_APPLIED;  
status_textBox.Text = ST_STEP_APPLIED;  
}  
// Atualiza barra de progresso e o tempo  
decorrido no controle UI  
AtualizarBarraProgr();  
  
break;  
  
case EstadoEnsaio.STATE_STEP_APPLIED:  
    if (stopwatch.ElapsedMilliseconds >=  
        tempo_medicao)  
    {  
        // Fim do ensaio de identificacao!  
  
        // Mover a maquina para o proximo estado!  
        estado_ensaio = EstadoEnsaio.STATE_END;  
        status_textBox.Text = ST_END;  
    }  
    // Atualiza barra de progresso  
    AtualizarBarraProgr();  
  
    break;  
  
case EstadoEnsaio.STATE_ABORT:  
    // Acoes tomadas quando o ensaio é abortado  
  
    // Parar contador  
    stopwatch.Stop();
```

```
// OBS: Aquisicao ja eh parada automaticamente,
// de acordo com a maquina de estados!

// Liberar pedal para o condutor
if (pParametros.EnviaCOM(PARAMETROS_ECU.
    ID_PEDAL_SIMULADO, pedal_min) == false)
{
    // Falha de comunicação! -> Abortar ensaio de
    // identificacao!
    estado_ensaio = EstadoEnsaio.STATE_COMFAILURE
        ;
    break;
}
if (pParametros.EnviaCOM(PARAMETROS_ECU.
    ID_MODO_PEDAL_SIMULADO, 0) == false)
{
    // Falha de comunicação! -> Abortar ensaio de
    // identificacao!
    estado_ensaio = EstadoEnsaio.STATE_COMFAILURE
        ;
    break;
}

// Desabilita timer para parar o ensaio!
timer10ms.Stop();

// Mostra mensagem de warning de teste abortado
MessageBox.Show("Ensaio de Identificação abortado
    !", "Ensaio abortado", MessageBoxButtons.OK,
    MessageBoxIcon.Warning);

// Retornar para o estado inicial
estado_ensaio = EstadoEnsaio.STATE_IDLE;
start_button.Text = "INICIAR"; // Reseta o texto
// do botao "Iniciar"
status_textBox.Text = ST_IDLE;

// Religa o controle da janela
this.ControlBox = true;

break;
```

```
case EstadoEnsaio.STATE_END:
    // Acoes tomadas no fim do ensaio de
    identificacao

    // Parar contador
    stopwatch.Stop();

    // OBS: Aquisicao ja eh parada automaticamente,
    de acordo com a maquina de estados!

    // Liberar pedal para o condutor
    if (pParametros.EnviaCOM(PARAMETROS_ECU.
        ID_PEDAL_SIMULADO, pedal_min) == false)
    {
        // Falha de comunicação! -> Abortar ensaio de
        identificacao!
        estado_ensaio = EstadoEnsaio.STATE_COMFAILURE
            ;
        break;
    }
    if (pParametros.EnviaCOM(PARAMETROS_ECU.
        ID_MODO_PEDAL_SIMULADO, 0) == false)
    {
        // Falha de comunicação! -> Abortar ensaio de
        identificacao!
        estado_ensaio = EstadoEnsaio.STATE_COMFAILURE
            ;
        break;
    }

    // Desabilita timer para parar atualizacao dos
    parametros
    timer10ms.Stop();

    // Mostra mensagem de sucesso!
    MessageBox.Show("Ensaio de Identificação
        realizado com sucesso!", "Ensaio finalizado",
        MessageBoxButtons.OK, MessageBoxIcon.
        Information);

    // Retornar para o estado inicial
    estado_ensaio = EstadoEnsaio.STATE_IDLE;
```

```
start_button.Text = "INICIAR"; // Reseta o texto
do botao "Iniciar"
status_textBox.Text = ST_IDLE;

// Religa o controle da janela
this.ControlBox = true;

break;

case EstadoEnsaio.STATE_COMFAILURE:
// Acoes tomadas quando o ensaio é abortado

// Parar contador
stopwatch.Stop();

// OBS: Aquisicao ja eh parada automaticamente,
// de acordo com a maquina de estados!

// Tentativa de liberar pedal para o condutor,
// pois houve falha de comunicação!
/*if (pParametros.EnviarCOM(PARAMETROS_ECU.
    ID_PEDAL_SIMULADO, pedal_min) == false)
{
    // Falha de comunicação! -> Abortar ensaio de
    // identificacao!
    estado_ensaio = EstadoEnsaio.STATE_COMFAILURE
    ;
    break;
}
if (pParametros.EnviarCOM(PARAMETROS_ECU.
    ID_MODAL_PEDAL_SIMULADO, 0) == false)
{
    // Falha de comunicação! -> Abortar ensaio de
    // identificacao!
    estado_ensaio = EstadoEnsaio.STATE_COMFAILURE
    ;
    break;
}*/

// Desabilita timer para parar o ensaio!
timer10ms.Stop();
```



```
// Mostra mensagem de warning de teste abortado
MessageBox.Show("Ensaio de Identificação abortado
                devido a falha de comunicação!", "Ensaio
                abortado - Falha de Comunicação",
                MessageBoxButtons.OK, MessageBoxIcon.Warning);

// Retornar para o estado inicial
estado_ensaio = EstadoEnsaio.STATE_IDLE;
start_button.Text = "INICIAR"; // Reseta o texto
                             do botao "Iniciar"
status_textBox.Text = ST_IDLE;

// Religa o controle da janela
this.ControlBox = true;

break;
    }
});
timer10msMutex.ReleaseMutex(); // Libera mutex
}
```