MOTIVAÇÕES PARA O *SOFTWARE* EMBARCADO E APLICAÇÃO DE DESENVOLVIMENTO ASCET NO GERENCIAMENTO DE UM MOTOR OTTO

Lucas Motta De Novaes[1], Armando Antônio Maria Laganá[1], Bruno Silva Pereira[1], João Francisco Justo Filho[1], Marco Isola Naufal[2], Marcos Henrique Carvalho Silva[1] e Paulo Alexandre Pizará Hayashida[1]

[1]Escola Politécnica da Universidade de São Paulo
[2]Instituto de Pesquisas Tecnológicas do Estado de São Paulo

E-mails: lnovaes.motta@usp.br, lagana@lsi.usp.br, bruno.sp@usp.br, jjusto@lme.usp.br, mnaufal@ipt.br, marcoshencarsil@gmail.com, paulo.hayashida@usp.br

RESUMO

A geração de *software* embarcado tornou-se um gargalo na introdução de novos produtos complexos no mercado, tais como: automóveis, aviões e plantas de controle industrial. A quantidade de *software* nos veículos atuais cresce exponencialmente. As forças motrizes desse desenvolvimento são a disponibilidade de *hardwares* mais baratos e poderosos e a demanda por inovações que ocasionam incremento de novas funcionalidades e estratégias para o controle eletrônico. Atualmente não existem modelos únicos para o desenvolvimento de *software* a serem seguidas, pois tanto o desenvolvimento do *software* embarcado, como o fluxo de suas variáveis são baseados em diferentes modelos e ferramentas (no domínio automotivo *Simulink* e ASCET são utilizados para representar projetos em diferentes níveis de abstração). No presente trabalho, é apresentado uma aplicação ASCET e o *design* de uma função de controle da mistura A/C em malha fechada em uma ECU dedicada ao desenvolvimento de *software* para um motor Otto de produção com 4 cilindros e 1600 cm^3, com validação dos testes em um banco dinamométrico. Para o acesso, controle e medição da operação da ECU utilizou-se a ferramenta INCA, a qual permitiu realizar as medições e calibrações dos resultados em tempo real.

INTRODUÇÃO

O padrão aberto OSEK/VDX – OSEK (*Offene Systeme und deren Schnittstellen für die Elektronik in Kraftfahrzeugen* – Sistemas Abertos e suas Interfaces para a Eletrônica em Veículos Motorizados) e VDX (*Vehicle Distributed eXecutive* – Execução veicular distribuída) foi fruto de uma convenção criada para estabelecer normas e especificações de *software* para serem utilizadas no desenvolvimento de unidades de controle eletrônico (ECU) embarcadas na década de 90 por um Consórcio entre a universidades e organizações automotivas de origem europeia. No decorrer desses 30 anos a quantidade de linhas de código dos *softwares* automotivos passaram de zero a dezenas de milhões. Um veículo *premium* atual, por exemplo, tem implementado cerca de 270 funções que apenas um usuário interage, implantadas em cerca de 70 módulos comunicantes [1], números estes que tendem a aumentar com o passar dos anos. O sistema de gerenciamento do motor é a parte mais significativa e valiosa em veículos modernos, sendo também uma das tarefas mais importantes no campo de controle automotivo. O sistema de controle do motor é um sistema embarcado de sinais mistos (contínuos e discretos), interagindo com o mundo físico através de sensores e atuadores. Comparado com

outros sistemas embarcados, requerem demandas de confiabilidade rigorosas (da ordem de 10^{-8} falhas/hora /carro) e compartilhamento de recursos e custos de maneira eficiente [2].

1. ESTRATÉGIAS PARA O DESENVOLVIMENTO EMBARCADO

Nesta seção, apresenta-se uma breve revisão da origem das padronizações e conceitos do desenvolvimento de *software* automotivo de maneira geral.

1.1. O esforço industrial para criação de arquiteturas para o *software* embarcado

A principal ideia é a visão de que as ferramentas de uma cadeia de processos de desenvolvimento podem ser interconectadas livremente e permitir uma troca contínua de dados. Os padrões definem protocolos, modelos de dados, formatos de arquivos e API's (*Aplicattion Programming Interface* - Interfaces de programação para aplicações) para o uso no desenvolvimento e teste de unidades de controle eletrônico automotivo. A ASAM (da tradução inglesa do alemão *Arbeitskreis Offene Systeme und deren Schnittstellen für die Elektronik in Kraftfahrzeuge*n – Sistemas Abertos e suas Interfaces para a Eletrônica em Veículos Motorizados) cooperou para o desenvolvimento do AUTOSAR (*AUTomotive Open System ARchitecture – Arquitetura do sistema aberto automotivo*), incorporando arquiteturas de *software*, padrões e arquivos de configuração para aplicativos e BSW(*Basic SoftWare*) , que permite interações entre os módulos do *software*. Estes módulos de aplicação são executados num RTE (*Run Time Environment* – ambiente de execução temporal) através de um VFB (*Virtual Functional Bus* – Barramento funcional virtual). Uma grande quantidade de ferramentas populares nas áreas de simulação, medição, calibração e automação de teste são compatíveis com as normas ASAM.

Para atender às demandas do desenvolvimento de sistemas de controle embarcado, a indústria automotiva desenvolveu a extensão padronizada do OSEK/VDX, que recebeu o nome de AUTOSAR. O AUTOSAR permite a especificação de componentes que implementam funções complexas, potencialmente distribuídas, em um veículo. Os principais objetivos de toda esta elaboração de métodos é proporcionar à indústria automotiva benefícios como modularidade, escalabilidade, transmissibilidade e reutilização de *software* entre projetos, variantes, fornecedores, clientes, sem a necessidade de reconfigurar, abrir e reconstruir o código e principalmente, aumentar o foco no desenvolvimento de novas estratégias de controle, sem perder estruturas já consolidadas.

1.2. Aspectos da engenharia de *software* embarcado

O desenvolvimento de *softwares* é um grande desafio devido à sua alta complexidade e inexistência física. Geralmente os problemas derivam de deficiências humanas, tais como: acurácia ineficiente ou capacidade limitada de visualização de interações complexas. A discussão científica para superar tais problemas foi denominada como engenharia de *software*. O código orientado foi pensado para eliminar as complicações iniciais de criação, e por isto, tem se tornado cada vez mais expressivo para o desenvolvimento de *softwares* [3].

O modelo tradicional do desenvolvimento de *softwares* de engenharia sugere uma realização em fases sucessivas. Na fase primária, os requisitos são analisados, então a

estrutura fundamental é projetada pela divisão em componentes de modelo. Esses componentes são implementados e testados individualmente antes de serem integrados e testados como um todo, para que em uma última fase seja realizada a adequação do *software* gerado a padrões normativos [4].

1.3. Model-Based Design

O *design* baseado em modelo emergiu como uma solução para os problemas de desenvolvimento de *software* integrado, testemunhados tanto pelo os esforços acadêmicos quanto os industriais. O princípio básico desta metodologia é afastar-se da codificação manual e de especificações informais de código, ao capturar requisitos incorporados de funcionalidades operacionais e não operacionais de modelos matemáticos abstratos. Claramente, um modelo matemático oferece um terreno comum para uma integração sistemática e coerente de diversos esforços na especificação do sistema, *design*, síntese (geração de código), análise (validação), execução (suporte de tempo de execução) e manutenção (evolução do *design*). Hoje, não existe nenhum modelo pronto que seja acordado para o desenvolvimento integrado de *software*, e os fluxos são baseados em diferentes ferramentas e modelos (e.g. no domínio automotivo, os modelos baseados em UML (*Unified Modeling Language*), Simulink e ASCET são comumente usados para representar projetos em diferentes níveis de abstração) [5].

A orientação de atuação divide as preocupações de funcionalidade (modeladas como protagonistas) das preocupações de interação de componentes (modeladas como *frameworks*) e fornece escopos bem definidos para aprimoramento do modelo e realização do sistema [2]. Este desenvolvimento como um todo se torna possível porque o SW automotivo é baseado no RTOS (*Real Time Operating System* – Sistema operacional de tempo real) automotivo (neste caso o RTA-OSEK). Em níveis de abstração, o *software* só acessa o RTA-OSEK e o *Basic Software* (BSW) que contém os *drivers* (*software* ao invés de *hardware*) de entrada e saída, além de *drivers* para funções específicas como GDI (*Gasoline Direct Injection* - injeção direta de "gasolina/combustível"). Isso permite que um AppSW possa ser desenvolvido focando-se apenas funções de controle, gerenciamento e no sistema operacional sem se preocupar com o alvo, tornando o *software* modular e independente.

2. **DESENVOLVIMENTO AUTOMOTIVO EM FASES SUCESSIVAS**

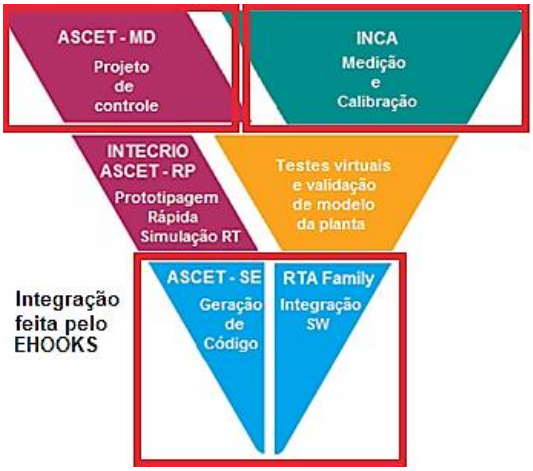
Nesta seção é descrito a metodologia de desenvolvimento de *software* embarcado e seus procedimentos característicos.

2.1. O ciclo V

O conceito referencial mais difundido para o desenvolvimento automotivo em geral é o ciclo V que visa a integração entre inspeção de qualidade e procedimentos de testes, especificando o cenário de componentes dentro de um determinado sistema. Um modelo V, de maneira geral, conforme a Figura 1, exemplifica os passos que devem guiar o desenvolvimento de um *software* automotivo, com o intuito de se empregar uma validação robusta do *software* de aplicação, este conceito tem sido amplamente utilizado na indústria automotiva, por propiciar segurança e qualidade na evolução de

aplicações automotivas, pois, compreende os cuidados a serem considerados em estratégias que deverão ser reproduzidas em massa.

Figura 1 – Ciclo V aplicado às implementações de *software* disponibilizadas pela corporação ETAS.



FONTE: Adaptado de [6].

Na Figura 1, foram destacados os passos do ciclo V utilizado nesta aplicação de projeto. Independentemente do processo clássico de *design* do sistema de controle ou do processo de desenvolvimento do ciclo V, o processo de projeto do sistema de controle é dividido em 4 etapas. A primeira etapa é o *design* funcional, como será abordado na seção seguinte.

2.2. Desenvolvimento funcional do controle

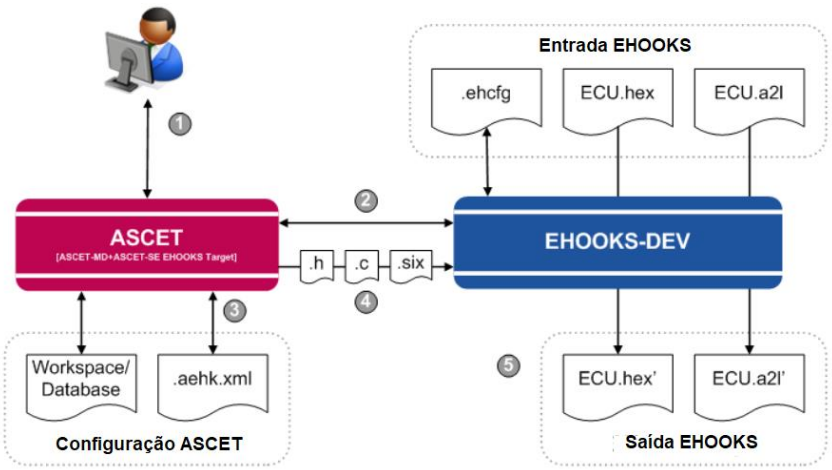
Com base em uma descrição conceituada do objetivo de controle, realiza-se a seleção dos algoritmos de controle baseados em modelos de planta de alto nível, uma plataforma de execução idealizada e requisitos de desempenho simplificados. As leis de controle geralmente são validadas por uma combinação de provas matemáticas e simulações de tempo contínuo, discreto ou de sinais mistos. O desenvolvimento do controle a ser introduzido na ECU, de maneira a considerar apenas o funcionamento lógico da estratégia trata-se do ponto de partida, onde não há a necessidade de consideração do *hardware* em que será implementado (este ponto é tratado posteriormente pelo *software*). Desta maneira, a estratégia de controle do AppSW pode ser desenvolvida de maneira abstraída do código fonte, ou seja, através das seguintes interfaces: diagrama de blocos, linguagem C, ESDL (*Embedded Software Description Language*) e/ou máquina de estados e, desta maneira, a integração do *software* é realizada automaticamente para o código objeto a partir destes diagramas e/ou linguagens.

Para a finalização desta fase, são realizados testes com entradas e saídas virtuais, que podem ser realizados em tempo real e/ou virtual. No caso do primeiro (*real-time*), existe a necessidade de um HW de prototipagem, que é um dispositivo dedicado para rodar o AppSW de controle e este, por ser dedicado, consegue rodar não só o AppSW como também o sistema operacional *real-time* que será utilizado na ECU final, neste caso, um sistema operacional automotivo chamado RTA-OSEK que é uma implementação dos padrões estabelecidos pelo consórcio OSEK/VDX.

2.3. Geração estruturada de código

A geração do código objeto a ser implementado na ECU de desenvolvimento, trata-se da compilação das linhas de código C e/ou blocos de diagramas lógicos desenvolvidos no ambiente de trabalho e na base de dados, onde as funções realizadas por essas linguagens de programação são associadas ao *hardware/software* alvo (que neste projeto, trata-se do LLSW da Flex-ECU). A compilação e associação dos componentes é implementada através do uso integrado do *software* E-HOOKS Dev. (Figura 2).

Figura 2 – *Workflow* da integração ASCET/E-HOOKS Dev.



1. Operador; 2. "*Link*"de integração; 3. ASCET – *Model Development* + ASCET – *Software Engineering*; 4. Extensões de código do AppSW; 5. Objetos gerados pelo E-HOOKS Dev. Fonte: Adaptado de [7].

O processo de compilação integrada resulta em dois arquivos essenciais fundamentados pela norma ASAM (*Association for Standardisation of Automation and Measuring Systems*), os arquivos ASAM-MCD-2MC (ASAP2) de extensão A2L e o código objeto extensão HEX. O formato A2L explicita nome de variáveis e respectivos endereços, fórmulas de conversão, e especificações da interface de *hardware*. O código hexadecimal (HEX) armazena as instruções de *software* e os dados utilizados na calibração, esta divisão é fundamental para garantir o pré-requisitos de rastreabilidade e segurança previstos dentro da engenharia de *software* embarcado.

2.4. Testes virtuais e virtualização

O processo do ajuste ideal de parâmetros para a ECU deve ser feito desde a primeira fase, onde se desenha e define as funções que vão gerenciar o sistema. Contudo, quanto mais distante da planta real e de um teste em malha fechada, mais complicado será definir os valores das variáveis de calibração e mais distantes estarão estas dos valores que terão ao finalizar a etapa de medição e calibração. Portanto, quanto antes o AppSW (*SoftWare* de aplicação), o SW e a ECU forem testados em *closed loop*, mais rápido se encontram e se resolvem os problemas. Além disso, é possível iniciar um trabalho de medição e calibração muito mais representativo desde a primeira etapa, tornando o desenvolvimento de uma unidade de controle mais eficiente, formando um laço de repetição condicional com base nos resultados desejados. Nesta etapa, se adota um

modelo da planta para que seja possível emulá-la em um PC com sistema operacional *real-time* que estimule a ECU com sinais elétricos idênticos aos existentes em uma conexão com a planta real, como se esta estivesse realmente conectada à planta, permitindo que se faça testes e validações de maneira completa sem o viés de perdas ou danos materiais oriundos de testes práticos. A partir da consideração deste conceito, o *Model-Based Design* permite interações entre suas etapas através de diversos recursos de ferramentas como: SiL (*Software-in-Loop*), MiL (*Model-in-Loop*), Prototipagem Rápida, *External By-pass* e *On-Target By-pass.*

2.5. Medição e calibração

A fase final do ciclo V corresponde à validação e teste na planta real (motor ou veículo), onde a ECU é conectada ao veículo real e o trabalho de calibração dos parâmetros é realizado. A calibração é feita acessando a ECU através de protocolos automotivos de medição e calibração como, por exemplo, acesso via CCP (*CAN Calibration Protocol*) ou ETK (*Emulator test probe (do alemão: Emulator-TestKopf*).
A utilização desses protocolos permite acesso direto à unidade de comando, para coleta dos conteúdos das variáveis de medição e para alteração de mapas e parâmetros de calibração de maneira *online* com o PC de trabalho. Desta forma é possível alterar mapas e variáveis enquanto a planta real (por exemplo, o motor) está em funcionamento, sem a necessidade de realização de uma nova gravação de *flash* da ECU. Nesta fase, ajustes relativos a combustíveis, sensores, dirigibilidade, economia de combustível e conforto podem ser realizados sem a necessidade da geração e compilação de um novo SW, desde que sejam previstos na confecção lógica do *software* de aplicação.

3. METODOLOGIA DA APLICAÇÃO

Nesta seção são abordados os métodos envolvidos no projeto de aplicação e suas origens, bem como os componentes de *hardware* e *software* utilizados. No desenvolvimento desta aplicação de projeto o ciclo V foi utilizado de maneira simplificada, com a utilização dos seguintes componentes: ASCET – MD, ASCET-SE, E-HOOKS Dev, RTA Family e INCA

3.1. Ferramentas de desenvolvimento de *software* embarcado

Esta seção apresenta as ferramentas utilizadas nas três fases estratégicas do desenvolvimento *model-based* que foram empregadas no desenvolvimento deste trabalho, sendo elas o desenvolvimento funcional, a geração do código estruturado e a fase de medição/calibração.

3.1.1. ASCET

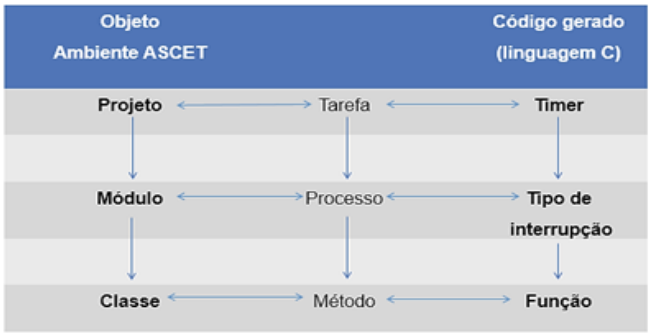
A ferramenta avançada de engenharia para simulação e controle ASCET (*Advanced Simulation and Control Engineering Tool*), segundo [3], foi concebida em 1997 para oferecer suporte às condições desafiadoras de mercado que motivaram a produção de *softwares* inteligentes de desenvolvimento.

ASCET oferece principalmente suporte ao paradigma do Modelo de Programação de Tarefas (MPT) para implementações de processador único de controladores de *software* incorporados. De acordo com o MPT, as especificações do sistema são dadas como uma coleção de componentes de

software denominadas *tasks* (tarefas) e uma política de agendamento apropriada. A especificação do sistema no ASCET é chamada de projeto e consiste em um conjunto de módulos que criam instâncias e definem processos. As classes são componentes reutilizáveis que consistem em variáveis e métodos. O estado do componente é mantido por variáveis. As saídas de componentes podem ser calculadas através de métodos de chamada. Os processos são métodos definidos em módulos e podem chamar os métodos definidos pelas classes alocadas dentro do módulo e se comunicam através de mensagens, que são variáveis compartilhadas protegidas por cópia local: cada processo funciona em uma cópia local de suas mensagens de entrada/saída. O mecanismo de segurança e proteção é garantido pelo sistema operacional subjacente (RTOS), que é assumido como compatível com OSEK. Os processos são agendados com tarefas escalonadas (planejadas) pelo RTOS, seja por base de tempo ou evento. O mecanismo de agendamento é baseado em prioridade estática com preempção [5].

Pode-se estabelecer uma relação de analogia entre a linguagem de alto nível gráfica do ASCET, sistemas operacionais de tempo real (que é como o programa foi idealizado para funcionar) e a linguagem de código C gerada em um nível mais baixo de código, exemplificada na Figura 3.

Figura 3 – Exemplo de correspondência entre os objetos no ambiente ASCET, RTOS (*Real Time Operating Systems*) e a linguagem de código C.



FONTE: O autor

As rotinas são executadas a partir do conceito *on target by pass* que consiste em um desvio de tarefas em função da rotina "alvo", tendo como principais vantagens a modularização temporal de rotinas, desconsiderando o efeito da integração entre elas e do compartilhamento do raciocínio lógico dos módulos, sem a necessidade da cessão e/ou compreensão de todo o projeto por terceiros.

3.1.2. INCA

O trabalho de calibração foi realizado através da ferramenta integrada de medição e calibração INCA(*INtegrated Calibration and Acquisition Systems*), permitindo tarefas como: administração dos dados de calibração e suas versões; uso de novos *softwares* sem necessidade de mudança de controladores ou *chips* de memória; conversão de sinais internos da ECU em tensões analógicas e vice-versa , medição de sensores que não estão diretamente conectados à ECU (disponíveis através do protocolo CAN); gravação do desempenho do

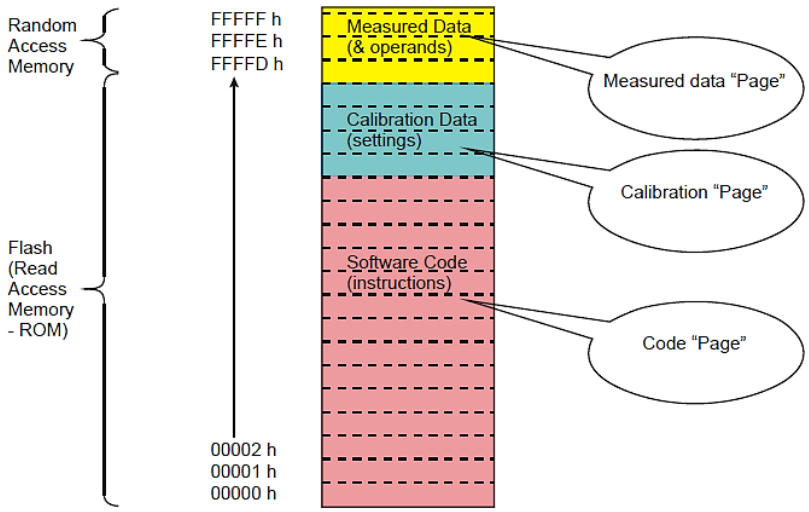
powertrain e avaliação de resultados. O ciclo típico de trabalho realizado neste *software* está representado na Figura 4.

Figura 4 - Ciclo típico para a calibração de um conjunto *powertrain.*

Fonte: Adaptado de [7].

As informações contidas dentro da ECU de desenvolvimento residem em um espaço de endereço próprio. O espaço de endereço é análogo a uma pilha, onde cada seção de correspondência é organizada sequencialmente, cada espaço da pilha contém um *byte* de informação, de modo que estas pilhas são agrupadas em páginas relacionadas com seus respectivos locais de endereços. Tipicamente, as unidades comando incorporadas para desenvolvimento que utilizam as normas automotivas possuem três páginas: *Calibration Data* – armazenada na página de calibração (*Calibration Page*) situada na memória *flash*; *Software* (instruções) – armazenada na página de códigos (*Code Page*) também situada na memória *flash*; *Measured Data and Operands* – armazenada na página de dados medidos (*Measure Data Page*) situada na memória RAM, como exemplificado na Figura 5.

Figura 5 – Estrutura de memória FlexECU – unidade voltada ao desenvolvimento de *software* no padrão automotivo.



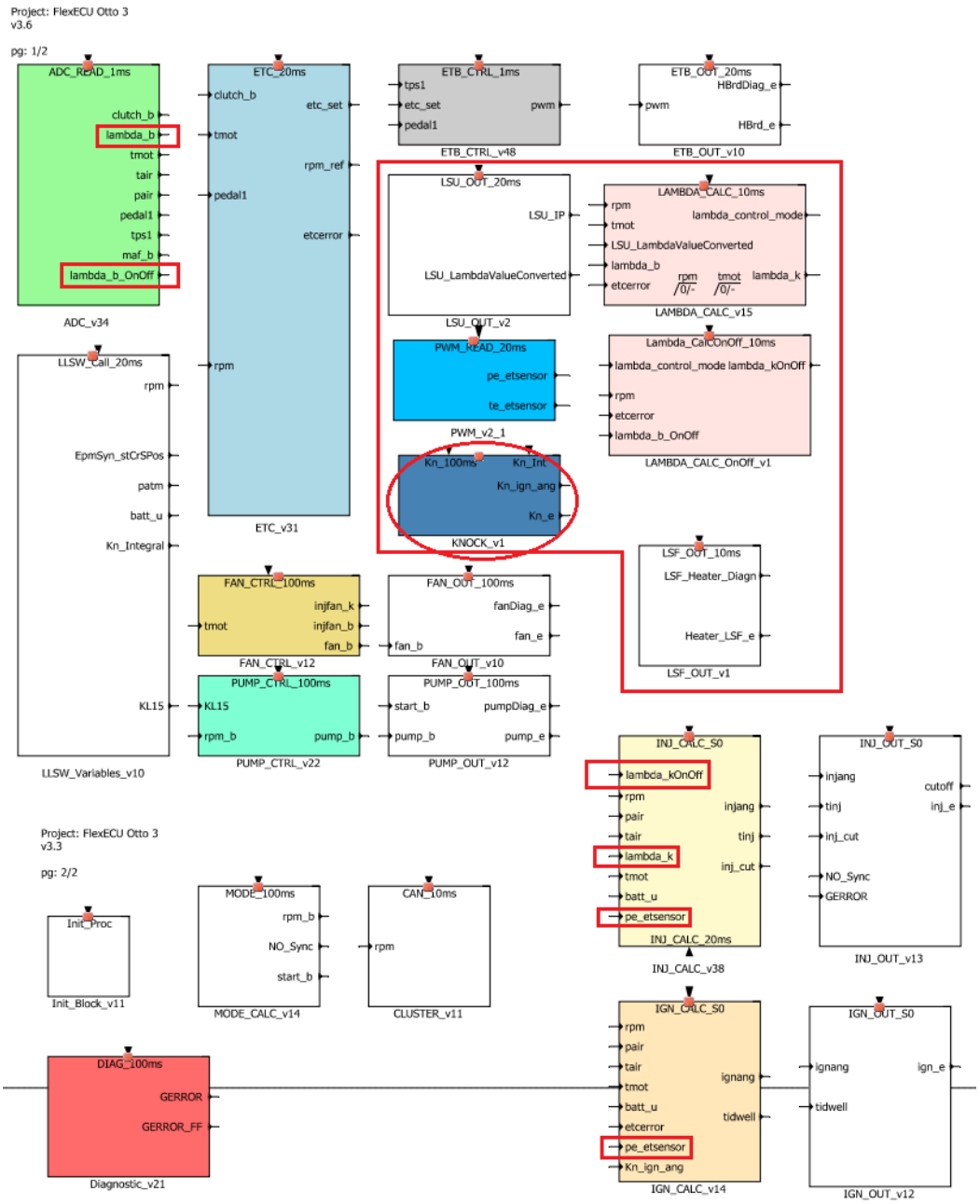
Fonte: Adaptado de [7].

As páginas de memória para calibração e medição de dados são lidas através do modo de acesso chamado ETK, este método de acesso ocorre de forma paralela, tendo como principal ganho a vantagem de não interferir nas ações do microcontrolador enquanto é realizado.

3.2. Projeto Otto III

Com a utilização do conceito ciclo V, foi desenvolvido para um motor 1.6L I4, um *firmware* de controle ou *AppSW*, denominado Otto III, utilizando a ferramenta ASCET no desenvolvimento do gerenciamento eletrônico, utilizando como *hardware* alvo a unidade de comando de desenvolvimento FlexECU em [8]. A Figura 6 exibe o projeto de *AppSW*, onde cada módulo possui funções e periodicidades distintas dentro do RTOS em que opera o *software* inteiro.

Figura 6 – Diagrama de representação dos módulos presentes no ambiente do *software* de aplicação ASCET com a versão de projeto Otto III.



As alterações adicionadas para o controle A/C (*lambda*) em malha fechada realizado neste trabalho, estão destacadas pelas marcações retangulares, com exceção do módulo destacado por uma elipse, que consiste em um bloco de detecção de detonação e correção do avanço de ignição, desenvolvido pelo coautor Paulo A. P. Hayashida. FONTE: O autor.

No desenvolvimento deste trabalho, o *firmware* de aplicação recebeu alterações necessárias para que fosse otimizado com módulos lógicos de controle necessários para a inserção do controle de estequiométrico do combustível A/C (*lambda*) em malha fechada. As dinâmicas e prioridades foram configuradas na organização do código em conformidade com a norma ASAM.

3.3. Descrição das aplicações de *Hardware*

Nesta seção são descritos os componentes de *hardware* e motor utilizados no presente trabalho.

3.3.1. Unidade de comando dedicada ao desenvolvimento

O *hardware* alvo neste trabalho é uma unidade dedicada ao desenvolvimento eletrônico de motores denominada Flex-ECU, a qual possui programação em código *open source*. Atribui-se o termo "flexível" ao nome da ECU devido à grande capacidade e abrangência de controle em seu *hardware*, concebido para simulação e desenvolvimento de estratégias automotivas. O *hardware* desta unidade é previsto para o comando de uma vasta gama de motores de combustão interna, sendo esta dedicada aos de ciclo Otto, que possuam subsistemas simples e/ou sofisticados, por exemplo, motores que contenham maior quantidade de cilindros, sobrealimentação (compressores e turbocompressores) e/ou injeção direta de combustível.

As especificações de *hardware* da plataforma de desenvolvimento descrita acima estão exibidas na Tabela 1.

Tabela 1 – Especificações do *hardware* da unidade de comando utilizada para o desenvolvimento ETAS/Bosch MEDC17 FlexECU *Gasoline* (dedicada aos motores SI – *Sparked Ignition*).

ETAS/Bosch FlexECU – versão "Gasolina", possui *hardware* projetado para motores de ignição por centelha (SI - *Sparked Ignition*, Otto)	
Microcontrolador	Infineon *Tri-Core* Tc1797 – 32 *bits*
Frequência de *clock*	180 MHz
Flash	1.8 MB Código; 215KB Dados
RAM	43KB
EEPROM	2KB (emulação)
Tensão de alimentação	8 – 16 V
Entradas digitais	8 (+2 dedicadas à IGN e *Flash*)
Entradas para frequência	4
Entradas para leitura de deslocamento do virabrequim e comando de válvulas	Hall
Entradas analógicas	30(11 passivas e 19 ativas)
Sensor de oxigênio (Sonda *lambda*)	2 LSU
Sensores de detonação (*Knock*)	Até 4
PWM (*Pulse Width Modulation*)	20 *Low-Side*;1 *High-Side*
Saídas digitais	11 Low-Side
Drivers ponte H	3
Drivers válvulas de fluxo de massa	2
Relés principais de energia	Externo
Drivers de ignição	Até 8 (bobina externa)
Drivers de injetores	Até 8 (GDI ou PFI)
Rede de comunicação	CAN (3 canais)
Protocolo	CCP (CAN *Calibration Protocol*)

FONTE: Adaptado de [9].

Toda a programação de acesso ao *hardware* à nível de máquina da FlexECU é feita de forma detalhada por um programa chamado LLSW (*Low Level SoftWare*), que não é acessível para edição, mas apenas para parametrização (Medição e controle). Nele também está contido o sistema operacional de tempo real RTOS chamado RTA-OSEK. A comunicação com o LLSW é feita a partir da utilização de API's, estas funções realizam chamadas para leitura de sinais e envio dos parâmetros necessários para o acionamento dos atuadores. O *software* de aplicação (ASCET) realiza o controle lógico do LLSW e seus respectivos *drivers.*

3.3.2. Descrição do motor utilizado

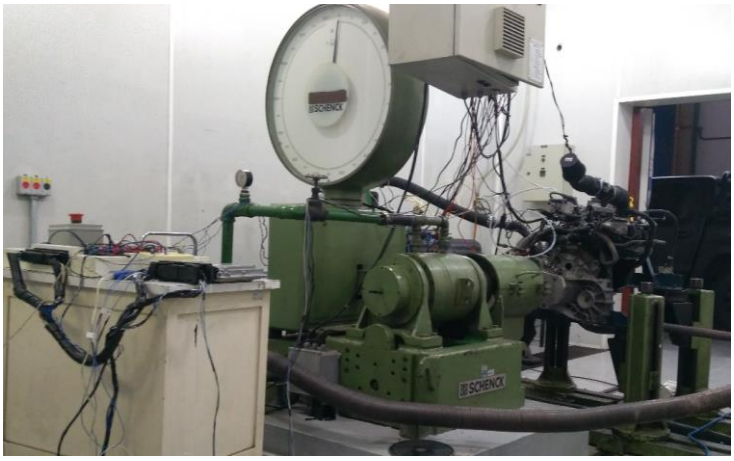
Os experimentos foram realizados em um motor naturalmente aspirado da família EA-111 VHT com 1.6L de deslocamento com disposição cilíndrica I4. As demais especificações deste motor estão listadas na Tabela 2.

Tabela 2 – Especificações técnicas do motor utilizado na aplicação dos testes.

Nomenclatura	EA - 111 VHT (CCRA) – etanol e/ou gasolina
Quantidade de cilindros	4 (quatro) dispostos em linha
Quantidade de válvulas	8 (oito), 2 (duas) por cilindro
Eixo de comando	SOHC (comando de válvulas simples situado no cabeçote)
Taxa de compressão	12,1:1
Deslocamento volumétrico	1598 cm^3
Potência Máxima (a nível do mar)	104cv (E) e 101cv (G) @ 5250 RPM
Torque Máximo (a nível do mar)	15,6 kgfm (E) e 15,4 kgfm (G) @ 2500 RPM
Diâmetro x Curso do êmbolo	76,5mm x 86,9 mm
ECU Original	Multiponto Bosch ME 7.5.30
Modo de injeção	Indireta multiponto sequencial 1-3-4-2 (MPFI)
Modo de ignição	Acionamento banco a banco – 2 (duas) bobinas de ignição dupla, acionando simultaneamente os cilindros 1-4 e 2-3 (centelha perdida)
Rotação máxima	6500 RPM

Os experimentos foram executados com o motor citado acima devidamente instalado em um banco de testes dinamométricos Schenck D360-1E de atuação hidráulica com sistema de automatização de controle eletrônico sp tronic ECAT situado no Instituto de Pesquisas Tecnológicas (IPT) do estado de São Paulo. A Figura 7 exibe o motor descrito acoplado ao dinamômetro empregado nos testes.

Figura 7 – Vista do motor utilizado acoplado ao dinamômetro hidráulico Schenck modelo D360-1E instalado na sala 1 do Laboratório de motores do Instituto de Pesquisas Tecnológicas de São Paulo.



Fonte: O autor.

3.4. Métodos utilizados na obtenção da medida *lambda*

Nesta seção, são descritos os métodos utilizados para obtenção do fator *lambda* da mistura A/C. Três métodos são estudados: Obtenção indireta através de *hardware* externo, leitura de uma sonda de banda estreita e leitura de uma sonda de banda larga UEGO LSU 4.9. A função de leitura foi desenvolvida em ambiente ASCET.

3.4.1. Utilização inicial de uma interface de testes Lambda

Utilizou-se a interface LA4 (*Lambda Analyzer Version 4 – Analisador Lambda versão 4)* para leitura de uma sonda de banda larga amplificada UEGO (*Universal Exhaust Gas Oxygen)* ou na designação alemã LSU (*Lambda Sonde Universal*) versão 4.9. A função do LA4 é controlar a referência de oxigênio da sonda para medição, com base na relação estequiométrica do combustível utilizado e efetuar o controle de aquecimento do sensor, o dispositivo LA4 é apresentado na Figura 8.

Figura 8 – Analisador *lambda* LA4

Fonte: O autor.

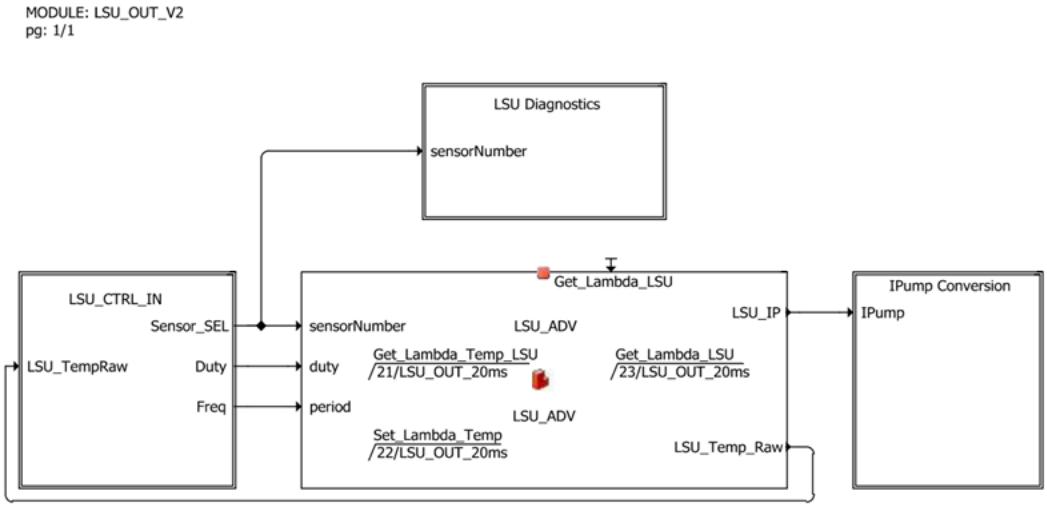
A relação de equivalência *lambda* pode ser diretamente monitorada via *display*, *assim* como, a corrente de bombeamento de oxigênio proveniente do sensor. O dispositivo possui uma saída analógica que viabiliza um sinal de tensão linear correlacionado à relação *lambda* medida, onde é possível

realizar ajustes de ganho e *offset*. O sinal do LA4 é monitorado na ECU de desenvolvimento via conversão A/D (Analógica/Digital*).*

3.4.1.1. Obtenção da medida da corrente de bombeamento em um sensor de banda larga amplficada

A Flex-ECU possui internamente em seu invólucro, um *hardware* para medição e controle de sensores de oxigênio *wide-band*, o circuito integrado (CI) CJ 125. O aquecimento da sonda também pode ser controlado pela ECU através da modulação PWM aplicado ao aquecedor da sonda. A utilização deste *hardware* interno torna possibilita tornar o projeto independente do analisador *lambda* LA4 no projeto. Foi desenvolvido um módulo de *software* para efetuar o aquecimento da LSU, controle, comunicação e leitura da corrente de bombeamento medida pelo CI CJ 125 e diagnose do mesmo. O mesmo é ilustrado pela Figura 9.

Figura 9 – Módulo LSU_OUT_V2 criado para leitura da corrente de bombeamento, controle do aquecimento do sensor de banda larga e diagnose do CI CJ 125.



Fonte: O autor.

O bloco central denominado LSU_ADV trata-se de uma classe criada para coordenar 3 (três) funções API's que realizam as rotinas de leitura da temperatura do sensor (Get_Lambda_Temp_LSU), acionamento de seu aquecedor integrado (Set_Lambda_Temp) e da corrente de bombeamento de oxigênio medida sem tratamento (Get_Lambda_LSU).

A hierarquia LSU_CTRL_IN trata a medida LSU_Temp_Raw e ajusta a temperatura do sensor através de uma malha PID na qual é configurado o *setpoint* desejado, controlando também a frequência e o *duty cycle* do PWM para o aquecedor integrado da célula de medida da sonda de banda larga amplificada.

A segunda hierarquia denominada LSU Diagnostics realiza as leituras das funções de diagnose do CJ 125, disponibilizando o status dos pinos de medida do sensor e pinos do de acionamento PWM do aquecedor.

A terceira e última hierarquia Ipump Conversion recebe o valor da corrente de bombeamento colhida do sensor de oxigênio através da classe LSU_ADV e o converte em relação de equivalência *lambda* a ser utilizada como referência para o controle A/C (*lambda*) em malha fechada.

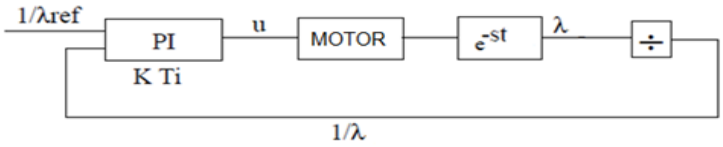
3.4.2. Monitoramento da tensão Nernst em um sensor de banda estreita

O sensor de oxigênio de banda estreita não requer cuidados tão complexos como o sensor de banda larga amplificada, por esta razão, é possível medir a relação de equivalência *lambda* diretamente da célula Nernst do sensor, medindo sua tensão de saída através de uma entrada analógica da *Flex-ECU*, considerando apenas o tempo de amostragem e resolução de sinal necessários.

3.5. Rotinas de controle

Nesta seção, são apresentadas as duas malhas utilizadas para controle da mistura A/C em malha fechada. Estas utilizam a mesma estrutura PI, porém, possuem ajuste dos ganhos através de critérios diferenciados. A Figura 10 exibe a estrutura PI utilizada em diagrama de blocos.

Figura 10 – Estrutura para realização do controle da relação *lambda* para um sistema que utiliza uma única sonda de banda larga em malha fechada.



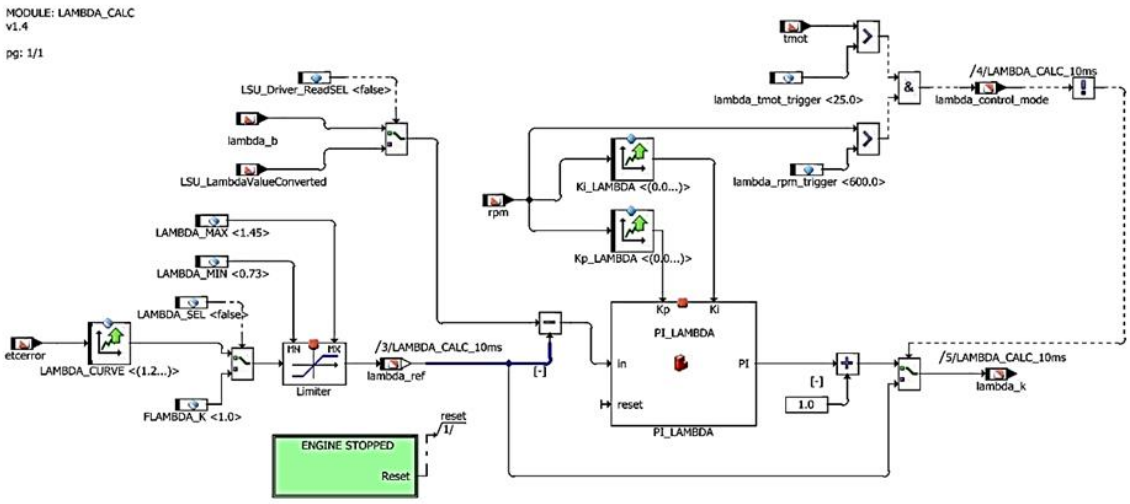
Fonte: Adaptado de [10].

Para definição dos ganhos dos sistemas que utilizam sensor de oxigênio de banda larga e banda estreita, aplicou-se as devidas identificações através de uma aproximação de 1ª ordem da resposta ao degrau da referência *lambda*, como proposto em [11]. Através dessa identificação, foi desenvolvido no ambiente ASCET, e com a metodologia proposta dois controladores PI como o proposto em [12], um utilizando um sensor de oxigênio de banda estreita e outro projeto distinto utilizando um sensor de banda larga.

3.5.1.1. Diagramas dos módulos de controle A/C (lambda)

Nesta seção, são discutidos os módulos de controle desenvolvidos e demonstra-se a empregabilidade dos recursos desenvolvidos. A Figura 11 ilustra o controle que utiliza sensor de banda larga.

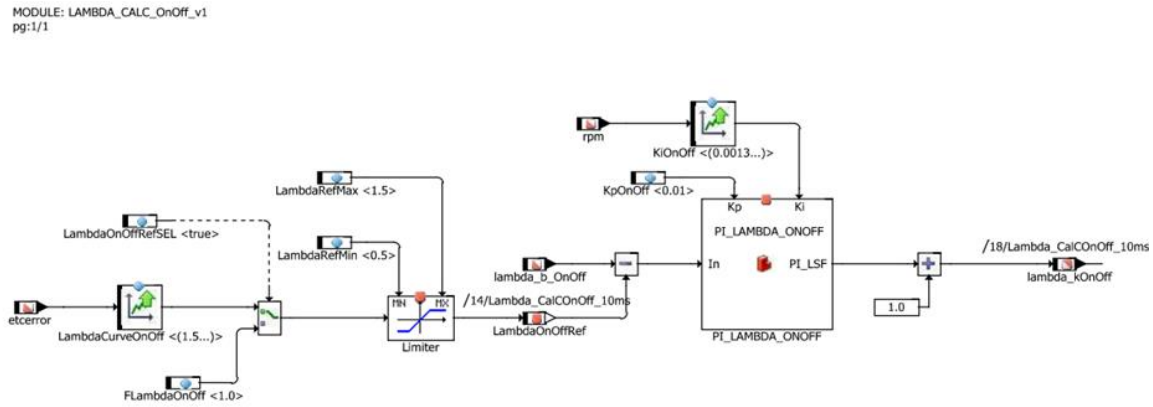
Figura 11 – Diagrama do módulo de controle LAMBDA_CALCv1.4 desenvolvido para medidas de sondas de banda larga.



Fonte: O autor

A rotina para sonda de banda larga possui ainda um seletor LSU_Driver_ReadSEL, onde é possível selecionar a leitura proveniente do analisador lambda (LA4), ou da conversão realizada através da corrente de bombeamento disponibilizada pelo controle do CI CJ125. A Figura 12 ilustra o controle com sensor de banda estreita.

Figura 12 – Diagrama do módulo de controle LAMBDA_CALC_OnOff_v1 desenvolvido para medidas de sondas de banda estreita.



Fonte: O autor.

Ambos os controles utilizam o valor de erro de rotação (variável etc_error) para determinar a referência *lambda* a ser utilizada. Os dois sistemas também utilizam a classe *Limiter* que tem a função de limitar o valor assumido por uma variável da relação de equivalência *lambda* medida, possuindo delimitadores definidos pelas entradas MN e MX.

3.6. Periodicidade das rotinas de controle

Segundo [12] o período de amostragem mínimo varia com a rotação do motor, em conformidade com a Eq. (1).

$$T_s = T_{ciclo} = \frac{120}{N} \quad (1)$$

onde N corresponde a velocidade do motor em RPS (Rotações por segundo).
As rotinas de controle do *software* são repetidas periodicamente, pois funcionam dentro de um RTOS, assim, as rotinas de controle *lambda* precisam possuir periodicidades fixas pré-definidas (selecionadas em *rasters)*, as quais foram definidas em 10 ms, ou seja, um tempo de amostragem suficiente até para rotação de corte do motor 6500 RPM. Portanto, o período de amostragem mínimo é o mesmo da execução desta *task*. Para as rotinas de diagnose foram escolhidas as amostragens mínimas de operação definidas em 20 ms.

4. **RESULTADOS**

Nesta seção, são discutidos os resultados obtidos para equivalência de leitura da tensão Nernst e da corrente de bombeamento pelo LA4 e CJ 125 e compensadores propostos.

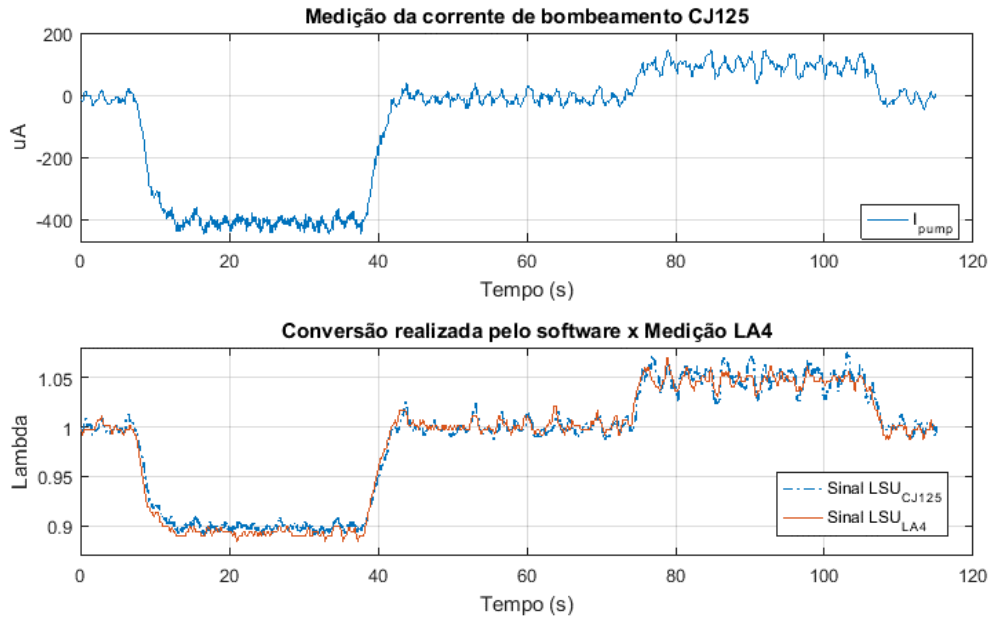
4.1. Obtenção das medidas da relação de equivalência *lambda*

Nesta subseção, comparou-se o controle da corrente de bombeamento desenvolvido em *software* e sua conversão como também o do monitoramento da tensão Nernst frente às conversões aplicadas pelo analisador *lambda* LA4.

4.1.1.1. Conversão da corrente de bombeamento

Este experimento teve o objetivo quantificar o delta entre a medida realizada pelo analisador *lambda* LA4 da medida originada da corrente de bombeamento de oxigênio medida do CI CJ 125 através do LLSW.

Figura 13 – Resultado do teste de comparação entre a medida da (a) corrente de bombeamento do CJ 125 e (b) a relação de equivalência λ convertida versus a medida do analisador *lambda* LA4 com sonda de banda larga.



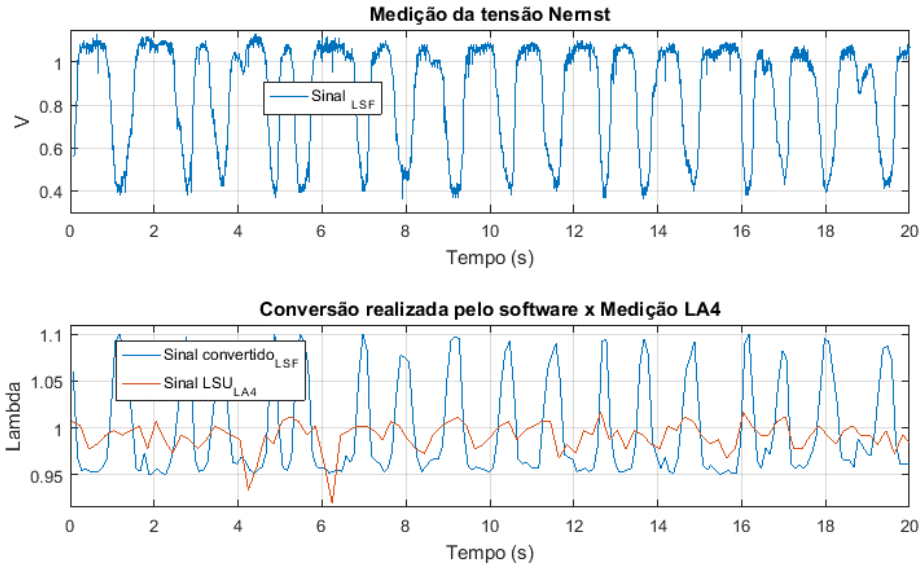
Fonte: O autor.

Ao observar Figura 13, nota-se evidentemente que a conversão da corrente de bombeamento se aproxima da medida do analisador *lambda* LA4 com um desempenho satisfatório, porém, também é possível notar que existe um mínimo desvio entre as medidas quando estas começam a oscilar, isso se deve ao fato de que o analisador *lambda* LA4 possui maior precisão, pois, o mesmo contém de maneira integrada, curvas de correção em função da relação estequiométrica e também das relações H/C e O/C do combustível utilizado.

4.1.1.2. Conversão da tensão Nernst

O seguinte experimento tem a função de repetir o teste realizado na subseção anterior, entretanto, a referência do controle A/C (*lambda*) foi fixada para uma relação de equivalência (λ=1), por se tratar de uma sonda de banda estreita vulgo binária, o controle preciso pode ser atingido apenas para mistura estequiométrica, ou seja, ($\lambda = 1.0$).

Figura 14 – Resultado do teste de comparação entre a medida entre (a) tensão Nernst da sonda de banda estreita e (b) a relação de equivalência λ convertida versus a medida do analisador *lambda* LA4 com sonda de banda larga.



Fonte: O autor.

A partir da análise da Figura 14, é possível perceber a razão da sonda de banda estreita ser comumente chamada também de sonda binária, possuindo apenas os estados de medida rico e pobre (a), constata-se que neste caso, a tensão Nernst deve estar sempre oscilando para alcançar-se a relação estequiométrica. Ao comparar-se o valor de conversão imposto no software com a medição real de um sensor de oxigênio de banda larga amplificada (b), nota-se que realmente o ponto estequiométrico ($\lambda = 1.0$), é atingido pela transição entre os estados rico e pobre.

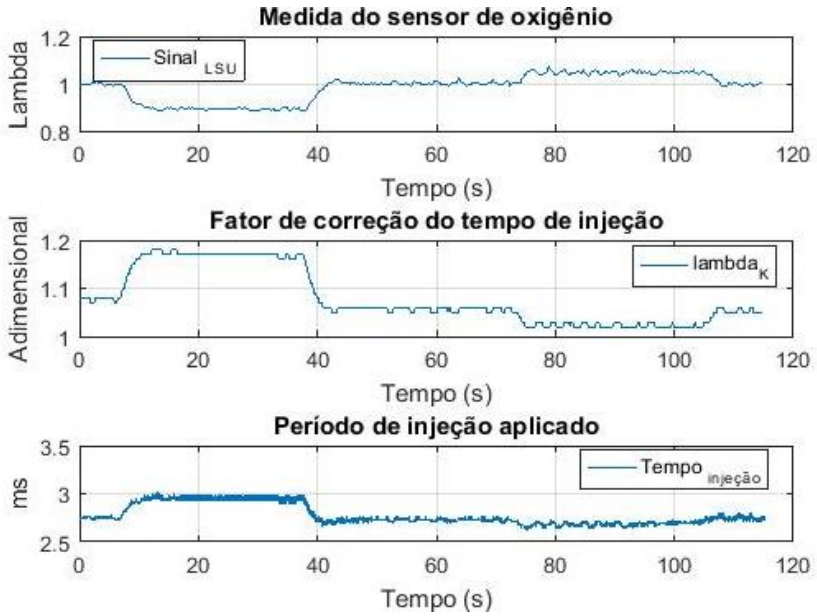
4.2. Controles A/C (*lambda*) em malha fechada

Nesta seção são relatados os testes dos controles da mistura A/C para os sensores de oxigênio de banda larga e banda estreita.

4.2.1.1. Sonda de banda larga

Para o presente ensaio o valor de referência é excitado com um degrau negativo (λ=0.9) no período de 10 a 40 segundos e depois com um degrau positivo (λ=1.05) de intervalo de 70 a 110 segundos. A correção do tempo de injeção também é analisada.

Figura 15 – Resultado do teste de transição da referência do controle da mistura A/C (lambda) (a) Resposta do sensor lambda de banda larga, (b) fator de correção do tempo de injeção e (c) tempo de injeção aplicado.



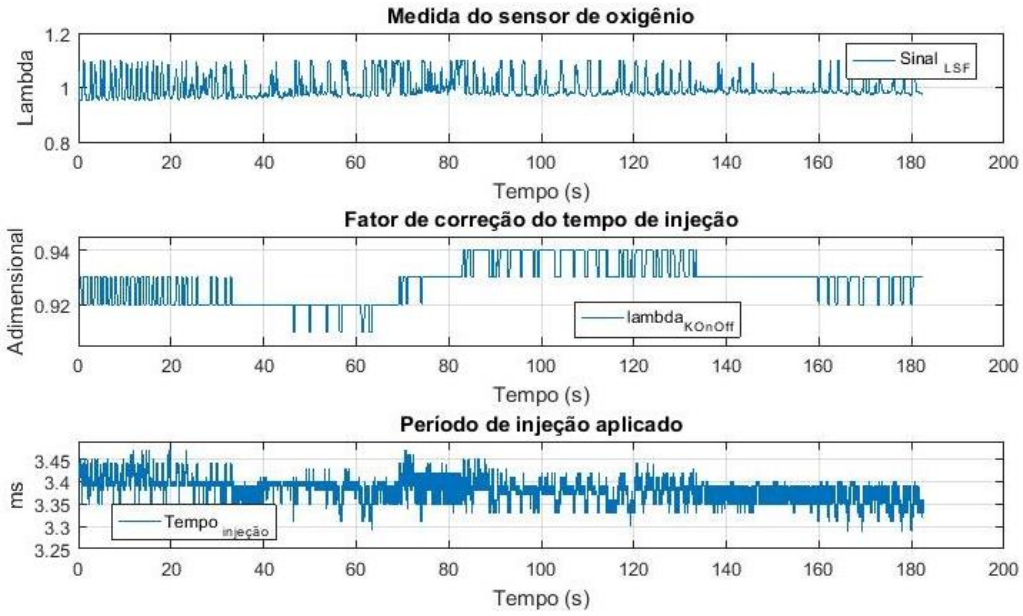
Fonte: O autor.

Pode-se observar através da Figura 15, que o resultado do teste de transição da referência do controle da mistura A/C (*lambda*) com a medida do sensor de banda larga amplificada (a), o fator de correção para o combustível injetado (b) e o período de injeção aplicado para abertura das válvulas injetoras (c). O valor da relação de equivalência λ (*lambda)* medida, corresponde a variação do valor de sua referência e se mantém estável durante os períodos de manutenção constante de sua referência.

4.2.1.2. Sonda de banda estreita

Para o presente ensaio o valor de referência é mantido fixo em (*lambda* = 1) e então é observado o comportamento do fator de correção e tempo de injeção. Os resultados são apresentados pela Figura 16.

Figura 16 – Resultado do teste de referência fixa do controle da mistura A/C (lambda), (a) Resposta do sensor de oxigênio de banda estreita, (b) fator de correção do tempo de injeção e (c) tempo de injeção aplicado.



Fonte: O autor.

Como pode analisar-se na Figura 16, períodos onde o sinal do sensor de oxigênio de banda estreita tem oscilações menores (a) é interpretado como mistura rica, logo o fator de correção (b) diminui o tempo de injeção (c). Já em períodos onde o sinal do sensor de oxigênio de banda estreita tem oscilações maiores (a) é interpretado como mistura pobre, logo o fator de correção (b) enriquece a mistura. É de relevância notar que o fator de correção neste caso varia no máximo em 3% do tempo de injeção (c), a fim de não alterar de forma brusca a velocidade e o funcionamento do motor.

CONCLUSÃO

O presente trabalho buscou apresentar de maneira geral, os principais aspectos e vantagens do desenvolvimento de *software* embarcado. A engenharia de *software* embarcado gerou soluções para potencializar a velocidade, qualidade e modularidade do desenvolvimento de *software* automotivo, construindo legados de maneira estruturada. Também foram apresentados projetos de aplicações que realizaram o controle da mistura A/C (*lambda*) em malha fechada para sondas *lambda* (sensores de oxigênio) com princípios de medida distintos. A vantagem de se desenvolver estratégias através da metodologia estudada foi demonstrada por meio da reutilização de componentes entre as funções, apesar das diferenças entre elas. O ambiente ASCET proporciona o emprego de outras extensões, de maneira a facilitar seu uso a partir do nível de abstração da finalidade a ser atingida. A ferramenta de aquisição INCA permitiu a calibração de todos os parâmetros necessários, bem como, a gravação dos dados de desempenho durante os experimentos.

REFERÊNCIAS

[1] M. Broy, K. I. H., A. Pretschner E C. Salzmann, "*Engineering Automotive Software*," *Proceedings of the* IEEE, vol. 95, nº 2, pp. 356-373, 2007.

[2] G. Huang, W. Huang E Y. Zhang, "*Actor-Oriented Methodology For Automotive Engine Control System Design*," *International Conference On E-Product E-Service And E-Entertainment*, pp. 1 - 4, 2010.

[3] M. Fuchs, D. Nazareth, D. Daniel E B. Rumpe, "BMW *ROOM*: An *Object-Oriented method for* ASCET," *Society Of Automotive Engineers*, pp. 1-11, 1998.

[4] J. Schäuffelle E T. Zurawka, *Automotive Software Engineering: Principles, Processes, Methods, and Tools*, Warrendale, PA: SAE *International*, 2005, p. 385.

[5] M. Baleani, A. Ferrari, L. Mangeruca E A. Sangiovanni-Vincentelli, "*Correct-by-Construction Transformations across Design Environments for Model-Based Embedded Software Development," Proceedings Of The Design, Automation And Test In Europe Conference And Exhibition*, Vols. %1 de %21530-159, 2005.

[6] H. Giese, G. Karsai, E. Lee, B. Rumpe E B. Schätz, *Model-Based Engineering of Embedded Real-Time Systems - International* Dagstuhl *Workshop*, Dagstuhl *Castle*, Germany: Springer-Verlag Berlin Heidelberg, 2007.

[7] ETAS INC. *INCA 7.1 - User Training Manual*. [S.l.]: [s.n.].

[8] P. D. C. Rossetti E J. P. F. D. Santos, "Otto III *by* FlexECU – Gerenciamento Eletrônico de um Motor VW 1.6L," São Paulo, 2015.

[9] ETAS, INC, Flex ECU-G1 Gasoline *Low Level Software Interface Definition Description* V1.2.2.

[10] L. Eriksson E L. Nielsen, *Modeling and control of engines and drivelines, First Edition* ed., T. Kurfes, Ed., Linköping *University*: John Wiley & *Sons* Ltd, 2014, p. 564.

[11] U. Kiencke E L. Nielsen, *Automotive Control Systems for Engine, Driveline, and Vehicle*, 2ª ed., Karlsruhe: Springer-Verlag Berlin Heidelberg, 2005, p. 512.

[12] B. S. Pereira, J. F. Justo E A. A. M. Laganá, "Controle Da Mistura Ar/Combustível Em Um Motor A Combustão: Sistema Em Malha Fechada," XXIII Simpósio Internacional De Engehnaria Automotiva, pp. 1-17, 2015.

AGRADECIMENTOS

Agradecemos a Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pela concessão de bolsas de mestrado e doutorado. Ao Instituto de Pesquisas Tecnológicas do Estado de São Paulo (IPT), por fornecer a sua infraestrutura e equipamentos para realização dos testes. A *Engineering Tool and Applications* GmbH (ETAS) do Brasil, por fornecerem as licenças de *software* e equipamentos de *hardware* utilizados.